Compiler Principles

Overview

Xiaoyuan Xie 谢晓园
xxie@whu.edu.cn
计算机学院E301

基本情况

课程类别：必修
课程学分数：3
课程学时数：54
前导课程：高级程序设计语言、离散数学、数据结构、计算机组成原理等
考核方式：笔试60%、作业和编程作业40%

教师信息

- 谢晓园，计算机学院 E301
  - xxie@whu.edu.cn
  - https://xiaoyuanxie.github.io/
- Tutor: 黎源
  - 1445660426@qq.com

设置目的

- 本课程是计算机科学与技术专业及相关专业的专业基础课，是一门理论与实践紧密结合的课程
- 开设本课程的目的是使学生了解并掌握编译过程中所涉及的基本理论和方法，具备分析和实现编译器的基本能力。

教学内容及学时分配

内　　容	学 时
1. 引论	4
2. 词法分析	6
3. 上下文无关文法	4
4. 自顶向下的语法分析	4
5. 自底向上语法分析	8
6. 语法制导翻译	8
7. 中间表示	4
8. 中间代码生成	6
9. 运行阶段存储组织与管理	6
10.代码生成及优化	4

教学内容及学时分配

- 引论：编译器的作用、工作过程、结构、构造方法等。
- 词法分析：正则表达式、有穷自动机、正则表达式到有穷自动机的转换和词法分析器生成工具Lex。
- 上下文无关文法：产生式、推导、语法树、二义性、文法设计。
- 自顶向下的语法分析：消除左递归与左公因子、递归下降语法分析、FIRST和FOLLOW集合、LL(1)文法、预测分析中的错误恢复。
- 自底向上语法分析：移进-归约技术；句柄与活前缀；LR(0)项与识别活前缀的自动机、移进-归约与归约-归约冲突、LR(0)文法、SLR文法、LL(1)项与自动机、LR(1)文法、LALR文法、二义性的处理、LR语法分析的错误恢复和语法分析器生成工具Yacc。

教学内容及学时分配

- 语法制导翻译：语法制导定义(SDD)、综合属性、继承属性、依赖关系图、语法树遍历与属性求值的关系、语法制导翻译规程、S属性与L属性、伴随语法分析过程的SDD实现。
- 中间表示：抽象语法树、三地址码、变量的作用域与符号表、类型表达式与类型检查。
- 中间代码生成：表达式的翻译、数组元素的引用、控制流的翻译、短路法与回填、过程调用的翻译。
- 运行阶段存储组织与管理：程序的运行与数据区、活动树与活动记录、栈式运行环境、嵌套过程的运行环境。堆管理与垃圾回收概述、参数传递方式及其实现；
- 代码生成及优化：目标语言、指令选择、寄存器分配、基本快与流图、基板快的优化、数据流分析初步、循环优化、指令流水线与指令调度。

学习资料

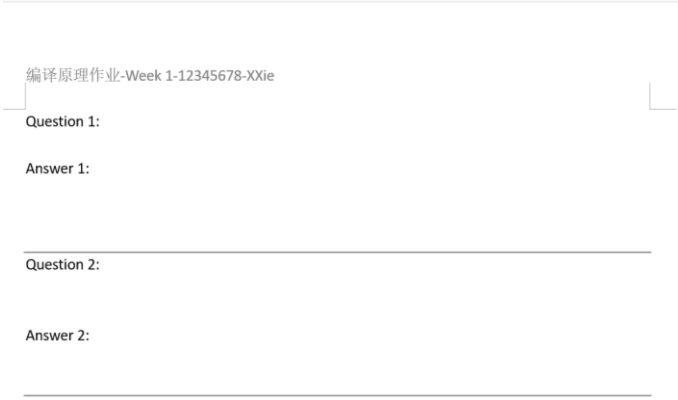
- 教 材
  - 编译原理 — 原理、技术与工具，机械工业出版社，[美]Alfred V·Aho， Monica S·Lam，Ravi Sethi，Jeffrey D·Ullman著. 赵建华、郑滔、戴新译，2008年12月
- 参考书：
  - 编译原理，高等教育出版社，陈意云，张昱，2003
  - 编译原理及实践，机械工业出版社，Kenneth C.Louden，冯博琴等译，2000.3
  - 现代编译器的Java实现（第二版），电子工业出版，Andrew W.Appel著，陈明译，2004.9
  - 程序设计语言 — 实践之路（第二版），电子工业出版，Michael L.Scott著，裘宗燕译，2007.6
  - Parsing Techniques - A Practical Guide (Second Edition), Dick Grune and Ceriel J.H. Jacobs, Springer

学习资料

- 课件
  - 每周一晚上上线（主页下载）
  - 同时上传上一周作业答案

作业

- 作业 – 每周
- word完成，命名规则：Compiler-Wi-id-Initial.docx
- 格式：


编译原理作业-Week 1-12345678-XXie

Question 1:

Answer 1:

Question 2:

Answer 2:

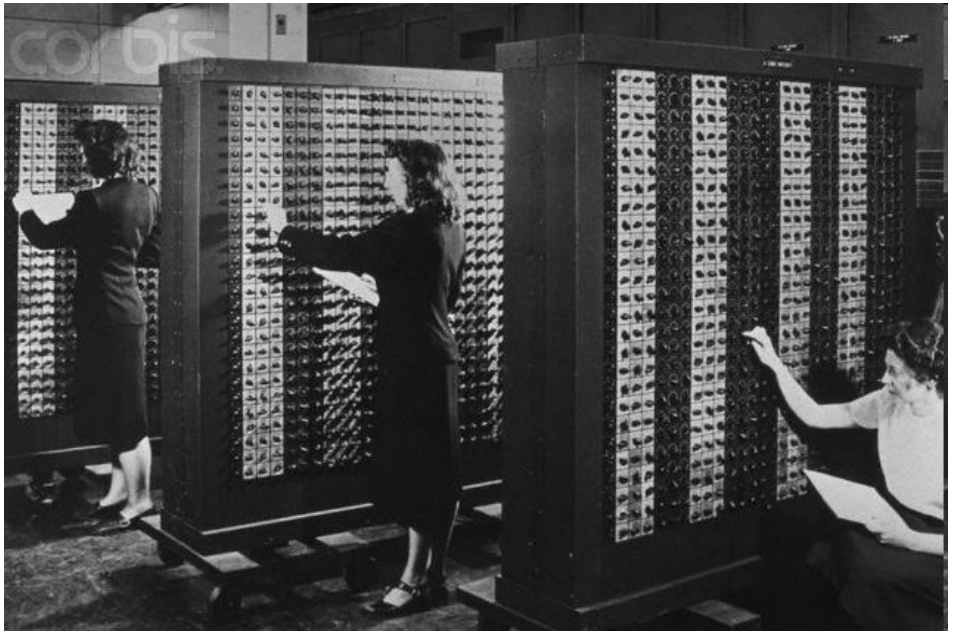
- Deadline：每周一上午课代表收齐，上午9点半之前发给课代表，过时不候

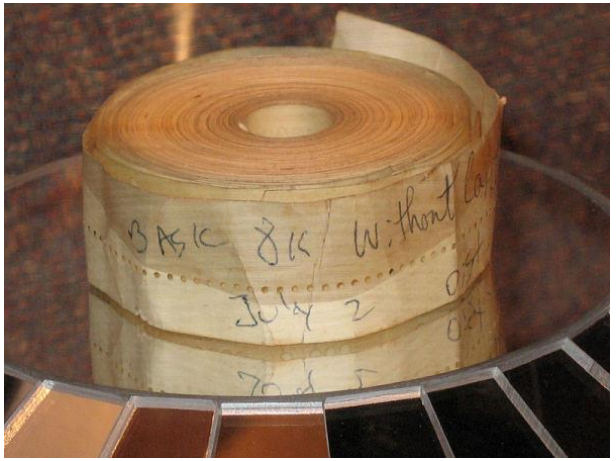
Lecture 1: Introduction

Xiaoyuan Xie 谢晓园
xxie@whu.edu.cn
计算机学院E301

How to Instruct a Computer?

- **最初，我们是这样的……**





How to Instruct a Computer?

- **问题：Programming bit-by-bit doesn’t scale**
- **Impossible:**
  - Programmer Productivity
  - Efficiency and Performance
- **我们需要high-level的编程语言**

How to Instruct a Computer?

- **Why high-level?**

President

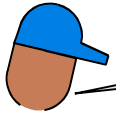
General

Sergeant

Foot Soldier

My poll ratings are low, lets invade a small nation

Cross the river and take defensive positions



Forward march, turn left Stop!, Shoot

How to Instruct a Computer?

- **High-level Candidature --- 自然语言?**
  - 具有强大的抽象能力，但是Ambiguous
  - Same expression describes many possible actions
- **Programming languages 程序设计语言**
  - high abstraction
  - precision (avoid ambiguity)
  - conciseness
  - expressiveness
  - modularity

1.1 程序设计语言

1.1 程序设计语言

发展史

史前文明：算盘，齿轮计算器，Jacquard Loom

1940~：机器语言

(a) Grace Hopper

(b) Punch card

Intel 机器码写的阶乘计算程序

```
10111000 00000001 00000000 00000000 00000000
10111010 00000010 00000000 00000000 00000000
00111001 11011010
01111111 00000110
00001111 10101111 11000010
01000010
11101011 11110110
11000011
```

1950~：汇编语言及汇编器

用文本表示机器语言

1. 机器指令用助记符表示.
2. 内存地址和指令地址用标识符表示.
3. 允许有注释.
4. 汇编器完成汇编语言到机器语言的翻译.

Intel~汇编语言的阶乘计算程序

```
;; 输入参数 N 放入寄存器EBX中,计算结果放入寄存器EAX中
Factorial:
        mov eax, 1      ;; 初始化输出result = 1
        mov edx, 2      ;; 初始化循环参数index = 2
L1:     cmp edx, ebx    ;; 如果 index <= N ...
        jg L2
        imul eax, edx   ;; result乘上index
        inc edx         ;; index递增1
        jmp L1          ;; 转移到循环始点
L2:     ret             ;; 返回
```

1.1 程序设计语言

发展史

1957：算数表达式的翻译

FORTRAN; COBOL; Igol60; LISP

FORTRAN --- FORmula TRANslator

- 与机器无关.
- 数学表达式，不需要计算机专业知识即可阅读和书写.
- 编译器完成数学表达式到汇编语言到翻译.

二次方程的求解

```
In FORTRAN :                ;In assembly language :
D = SQRT(B*B - 4*A*C)       mul t1, b, b     sub x1, d, b
X1 = (-B + D) / (2*A)       mul t2, a, c     div x1, x1, t3
X2 = (-B - D) / (2*A)       mul t2, t2, 4    neg x2, b
                            sub t1, t1, t2   sub x2, x2, d
                            sqrt d, t1       div x2, x2, t3
                            mul t3, a, 2
```

1960~：算法语言的诞生（递归与循环）

涌现了上百种程序设计语言 (特殊目的语言; 通用语言)

ALGOL (ALGOrithmic Language) --- 算法语言的诞生

- Backus-Naur 范式对语言形式描述.
- 递归调用，控制结构和调用方式 (传值与传名).
- 现代程序设计语言的雏形.

```
Iteration (loops):              Recursion:
r = 1;                          int  fact (int n)
for (i = 2; i <= n; i++)        { if (n <= 1)
  r = r * i;                        return 1;
                                  else return fact(n - 1) * n;
                                }
```

1.1 程序设计语言

发展史

1970~：数据结构的自动表示
简化, 抽象 (PASCAL; C;)

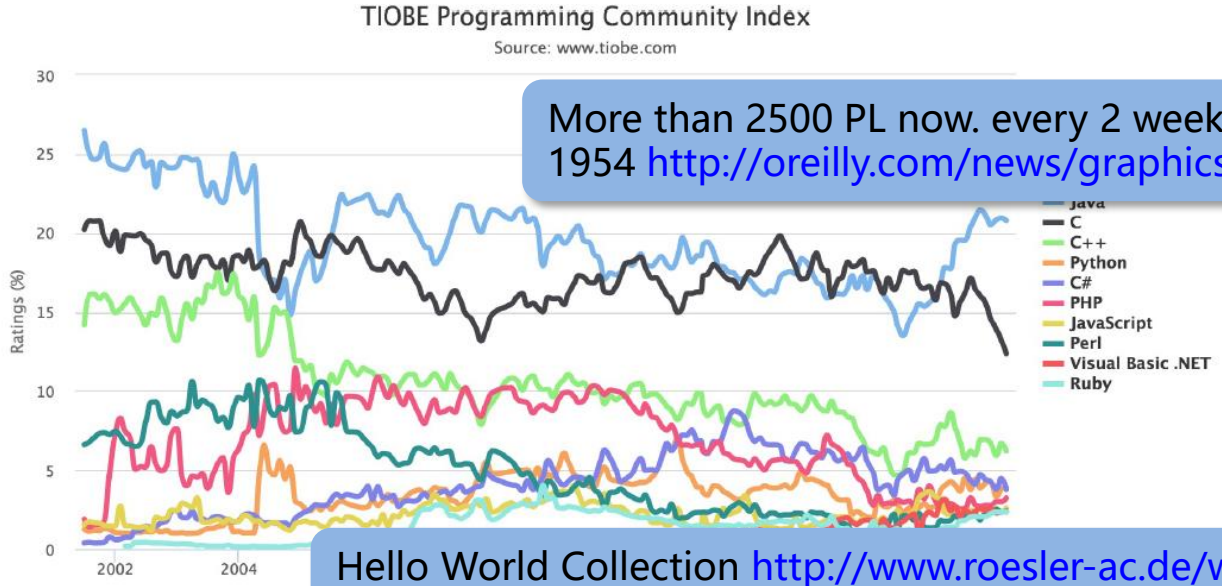
1980~：面向对象语言 (Ada; Modular; Smalltalk; C++)

1990~：网络语言 (Java), Libraries,
脚本语言 (Perl; Javascript)

2000~：说明语言(XML,UML,Z)

1.1 程序设计语言

使用统计



More than 2500 PL now. every 2 weeks, a new PL is born since 1954 http://oreilly.com/news/graphics/prog_lang_poster.pdf.

Hello World Collection http://www.roesler-ac.de/wolfram/hello.htm
收集了428 个不同语言写的``Hello World'' 程序.

1.1 程序设计语言

- **分类（几千种程序设计语言）**
  - 功能
    - 科学计算(Fortran); 商业数据处理(Cobol); 表处理(Lisp); 格式处理(Latex); 数据库语言(SQL);
  - 抽象级别
    - 低级：机器语言 & 汇编语言
    - 高级 (不同范例paradigms)
  - 划代(ith-Generation Language，iGL)
    - 1GL：机器语言
    - 2GL：汇编语言
    - 3GL：高级程序设计语言，如FORTRAN，ALGOL，BASIC，LISP等；
    - 4GL：为特定应用设计的语言，如数据库查询语言SQL,文本排版Postscript等；
    - 5GL：指基于逻辑和约束的语言，如Prolog，OPS5

1.1 程序设计语言

- **高级程序语言（不同范型：paradigms）**
  - 过程式(Procedural programming languages--imperative)
    - 程序中指明如何完成一个计算任务
    - FORTRAN, PASCAL, C
  - 函数式(Functional programming languages--declarative)
    - 程序中指明要进行哪些计算
    - LISP, HASKELL, ML, OCAML, SCALA…
  - 逻辑式(Logical programming languages--declarative)
    - 事实+推理规则
    - PROLOG
  - 对象式(Object-oriented programming languages)
    - 支持面向对象编程
    - Smalltalk, Java, C++, Eiffel, Ruby
  - 说明式语言(Declarative programming): 与上述命令式(Imperative language) 不同，没有控制结构，甚至没有赋值，仅有问题说明，或者 说纯数学定义

1.1 程序设计语言

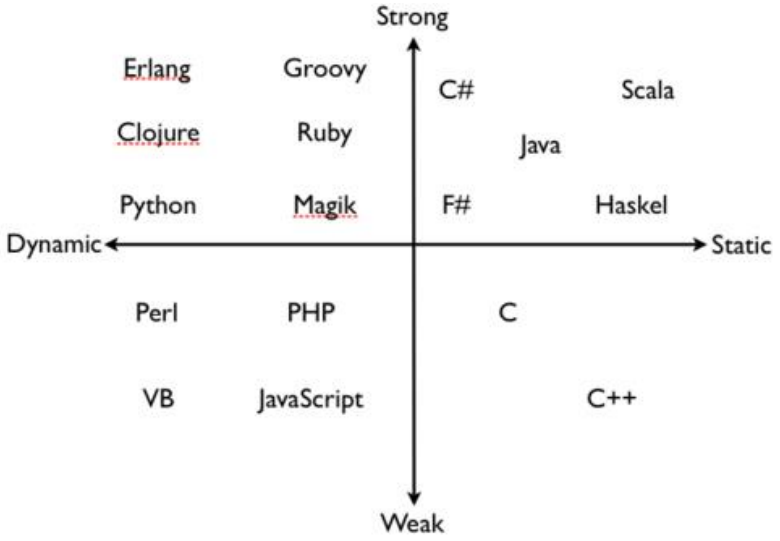
- **高级程序语言（不同转换方式）**
  - 编译型语言
    - 需通过编译器（compiler）将源代码编译成机器码，之后才能执行的语言。一般需经过编译（compile）、链接（linker）这两个步骤。编译是把源代码编译成机器码，链接是把各个模块的机器码和依赖库串连起来生成可执行文件。
    - 优点：编译器一般会有预编译的过程对代码进行优化。因为编译只做一次，运行时不需要编译，所以编译型语言的程序执行效率高。可以脱离语言环境独立运行。
    - 缺点：编译之后如果需要修改就需要整个模块重新编译。编译的时候根据对应的运行环境生成机器码，不同的操作系统之间移植就会有问题，需要根据运行的操作系统环境编译不同的可执行文件。
    - 代表语言：C、C++、Pascal、swift
  - 解释型语言
    - 不需要编译，相比编译型语言省了道工序，解释性语言在运行程序的时候才逐行翻译。
    - 优点：有良好的平台兼容性，在任何环境中都可以运行，前提是安装了解释器（虚拟机）。灵活，修改代码的时候直接修改就可以，可以快速部署，不用停机维护。
    - 缺点：每次运行的时候都要解释一遍，性能上不如编译型语言。
    - 代表语言：JavaScript、Python、Erlang、PHP、Perl、Ruby
  - 混合型语言
    - 比如C#,C#在编译的时候不是直接编译成机器码而是中间码，.NET平台提供了中间语言运行库运行中间码，中间语言运行库类似于Java虚拟机。.net在编译成IL代码后，保存在dll中，首次运行时由JIT在编译成机器码缓存在内存中，下次直接执行（博友回复指出）。
    - Java先生成字节码再在Java虚拟机中解释执行。

1.1 程序设计语言

- **不同的程序设计语言机制(函数式、过程式、逻辑式、对象式)，需要采用不同的技术编写编译程序**
  - 过程式语言的编译是对象式语言编译的基础
- **本课程重点关注*过程式程序设计语言*编译程序的构造原理和技术**

1.1 程序设计语言

- **动态类型语言（Dynamically Typed Language）**
  - 编译时不知道变量类型，运行时才决定
  - 类型错误属于运行错误，运行时报错
- **静态类型语言（Statically Typed L**
  - 编译时候决定变量
  - 类型错误属于语法错误，编译器报错
- **强类型定义语言（Explicit type co**
  - 偏向于不容忍隐式类型转换
- **弱类型定义语言（Implicit type co**
  - 偏向于容忍隐式类型转换



1.2 编译器概述

1.2 编译器概述

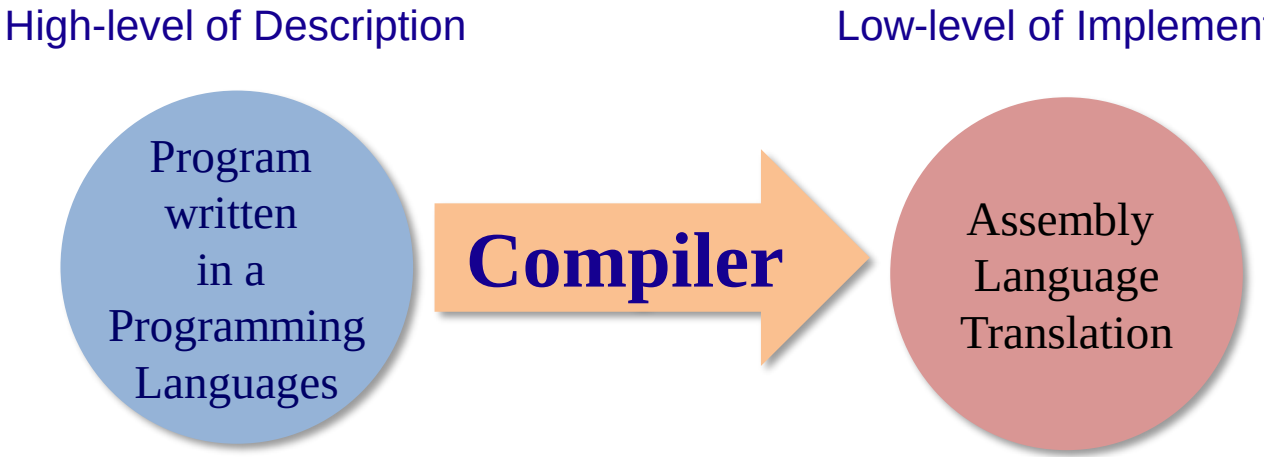
- 编译器(Compiler) 将某种语言(**源语言**)编写的程序翻译成**语义等价**的另一种语言(**目标语言**)编写的程序

- 目标程序若是可执行的机器语言程序，则可以被用户调用，处理输入并产生输出。

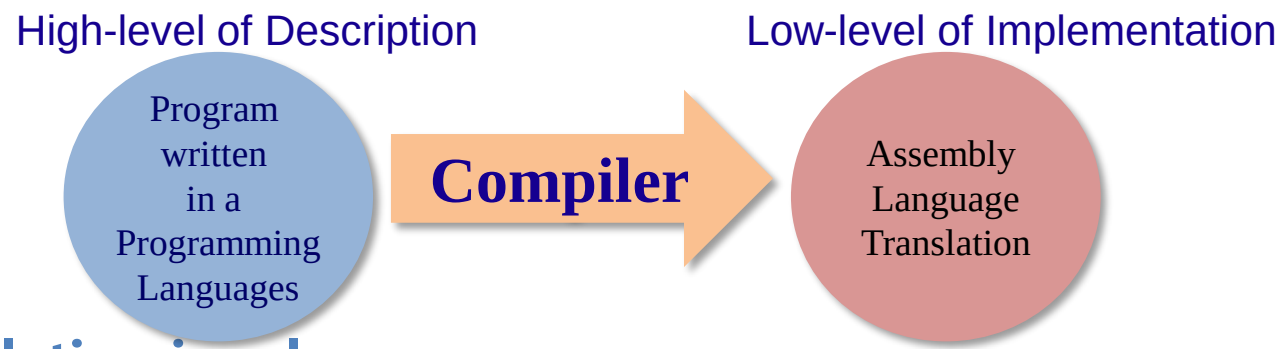
- 目标程序若是汇编语言的程序，则须经汇编器汇编后方可执行。
- 编译器的重要任务之一是报告它在翻译过程中发现的源程序中的错误。

1.2 编译器概述

- **编译器的重要使命：Translate a program**



1.2 编译器概述



- **Translation involves:**
  - Read and understand the program
  - Precisely determine what actions it require
  - Figure-out how to faithfully carry-out those actions
  - Instruct the computer to carry out those actions

1.2 编译器概述

- 回顾例子

President

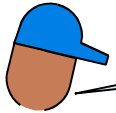
General

Sergeant

Foot Soldier

My poll ratings are low, lets invade a small nation

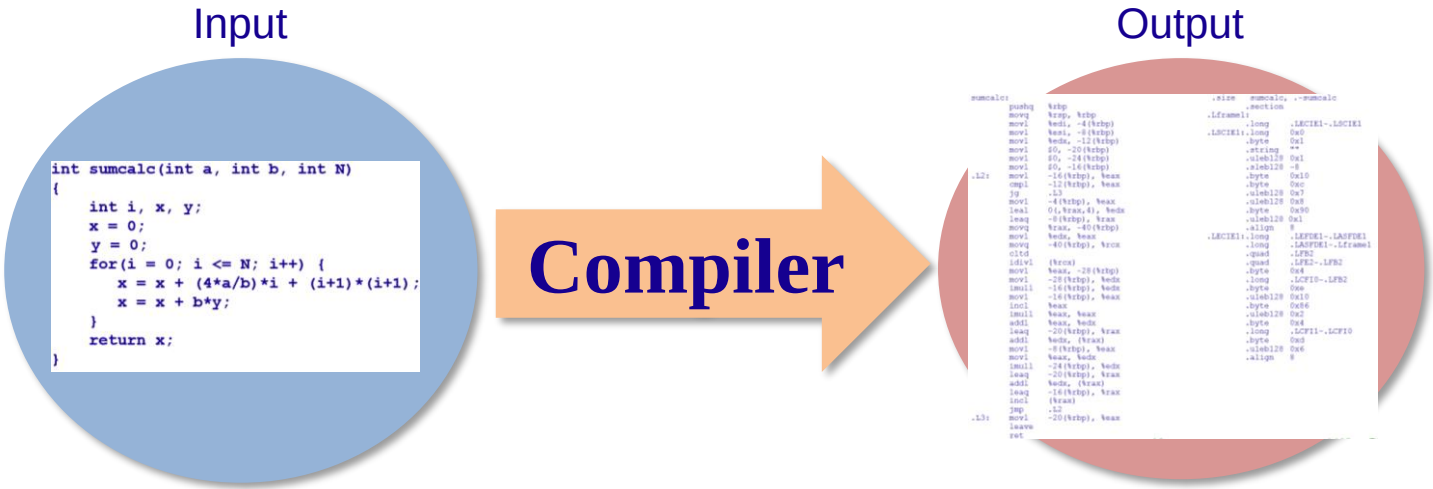
Cross the river and take defensive positions



Forward march, turn left
Stop!, Shoot

1.2 编译器概述

- 示例


Input
Output
Compiler
int sumcalc(int a, int b, int N)
{
int i, x, y;
x = 0;
y = 0;
for(i = 0; i <= N; i++) {
x = x + (4*a/b)*i + (i+1)*(i+1);
x = x + b*y;
}
return x;
}

1.2 编译器概述

- **输入：Standard imperative language (C, C++)**
  - State
    - Variables,
    - Structures,
    - Arrays
  - Computation
    - Expressions (arithmetic, logical, etc.)
    - Assignment statements
    - Control flow (conditionals, loops)
    - Procedures

1.2 编译器概述

- **输出：机器代码**
  - State
    - Registers
    - Memory with Flat Address Space
  - Machine code – load/store architecture
    - Load, store instructions
    - Arithmetic, logical operations on registers
    - Branch instructions

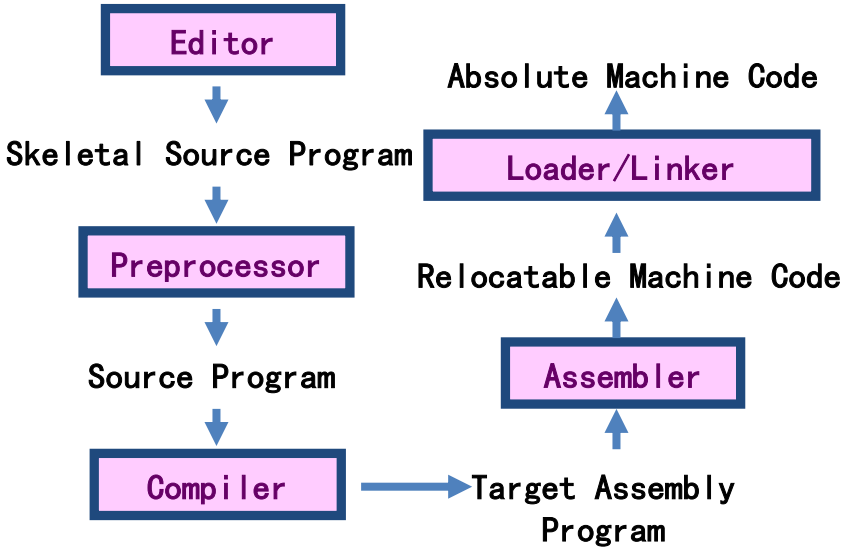
1.2 编译器概述

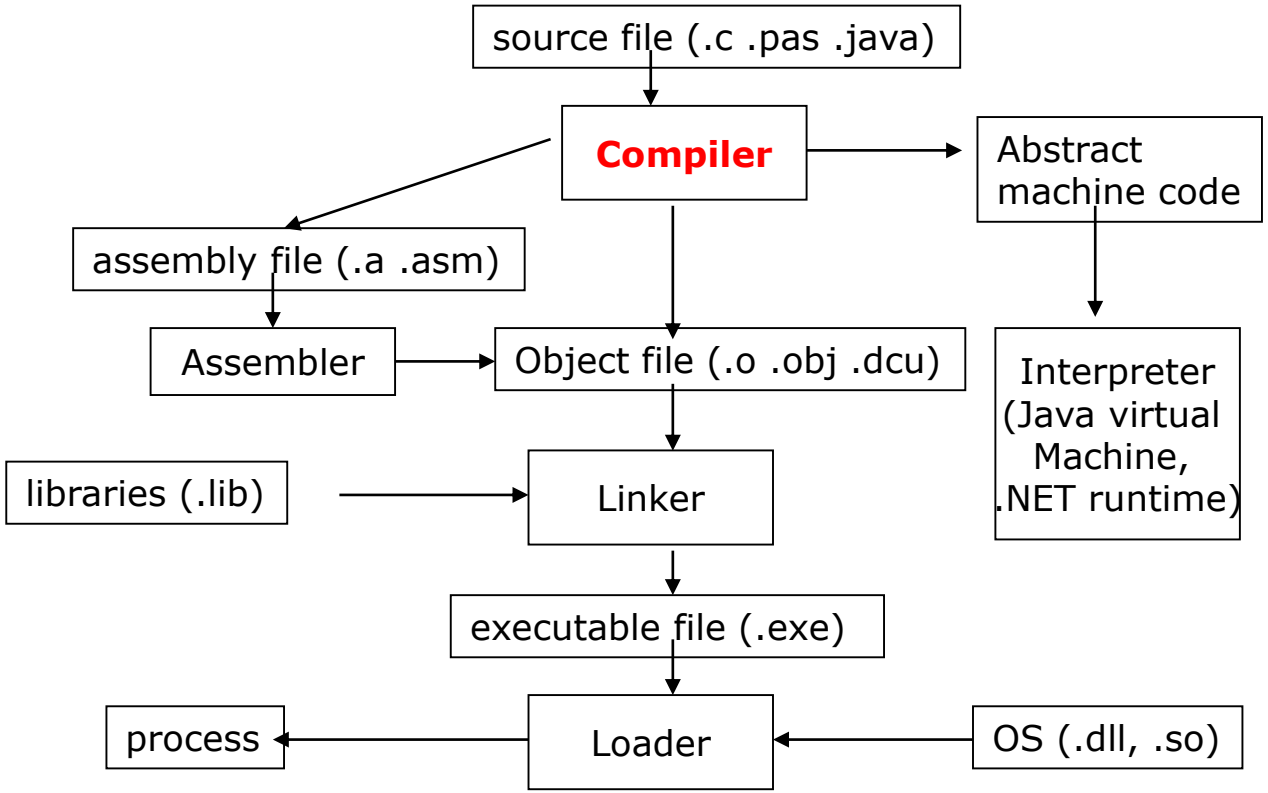
编译程序的伙伴程序

- 编辑器 (editor)　除一般的文本编辑功能外，还可以对正在编辑的文本进行分析、提示、自动提供关键字匹配等功能；
- 预处理器(preprocessor) 删除源程序中的注释、执行宏替换以及包含文件的嵌入等；
- 汇编程序(assembler) 将编译程序生成的汇编代码汇编成机器代码；
- 连接程序(linker)　将不同的目标文件连接到一个可执行的文件中；
- 装入程序(loader)　将程序加载到内存中以便执行；

1.2 编译器概述

- 整体流程




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)

1.3 编译器的组成

1.3 编译器组成

如何翻译

- 考虑 “自然语言翻译”过程：从中文到英文

你能够通过自己的努力实现你的梦想！

- 翻译的一般过程:

你能够通过自己的努力实现你的梦想！

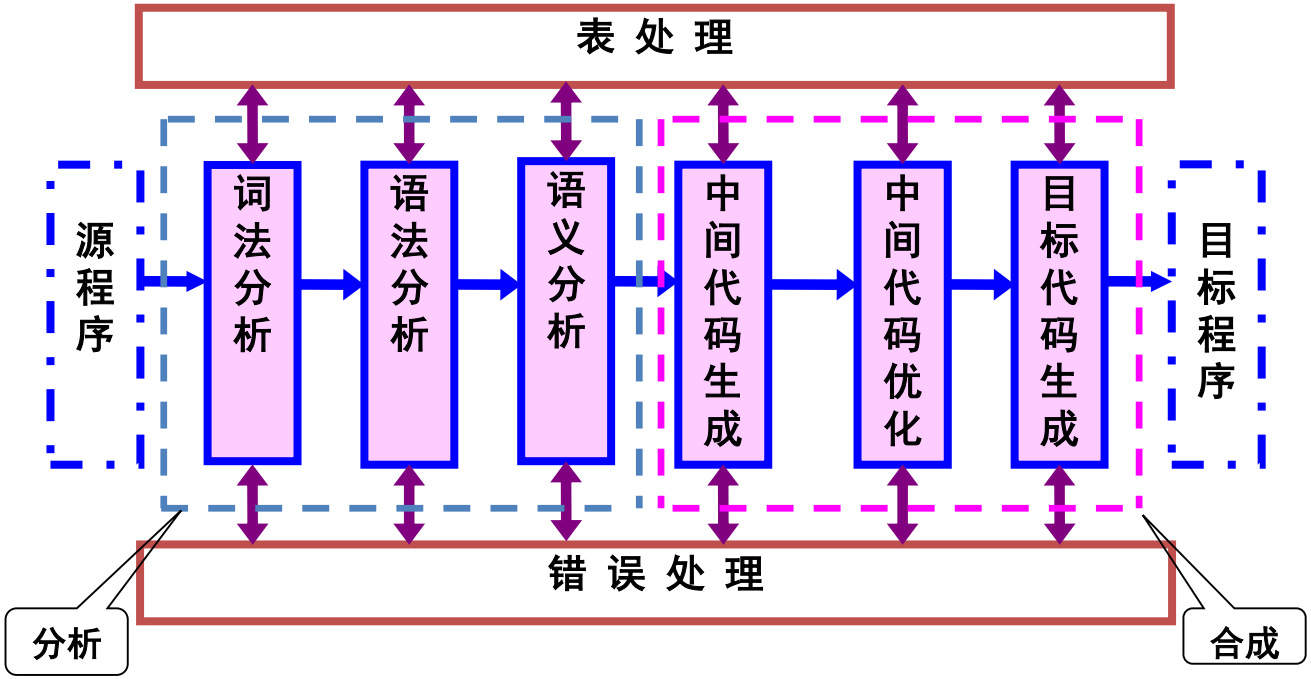
You can put your dreams into reality through your efforts!

1.3 编译器组成

- **自然语言翻译过程总结**
  - 掌握源语言和目标语言：词法、语法和语义
  - 翻译过程包括：
    - 分析源句子是否正确
      - 拼写，包括识别单词及其属性
      - 依据源语言的语法建立语法结构
      - 检查句子是否有意义

    - 将句子翻译成目标语言
      - 翻译每个语法部分
      - 将其组合成有意义的目标语言句子

1.3 编译器组成



1.3 编译器组成

- **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），是于机器无关的部分
- **合成部分（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），是于机器相关的部分

1.3 编译器组成

- **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

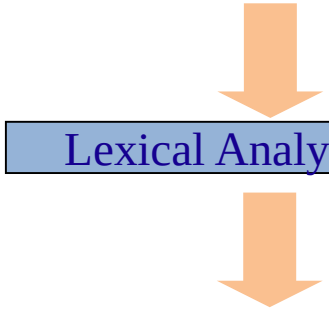
1.3 编译器组成 --- 词法分析

- **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： <token-name, attribute-value>
    - 程序语言处理的最小单位
    - 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
  - 程序语言规定了单词构成的规则和单词类别.

Example of C

- operators, separators: ! % + - ++ -- >> == ;
- identifiers, keywords.
- constants: 'A', "ABC", 0XAA, 0XAAL.

1.3 编译器组成 --- 词法分析



报错：18..23 + val#ue

Variable names cannot have ‘#’ character

Not a number

1.3 编译器组成 --- 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）
  - sentence是对单词的再次重组
  - 程序设计语言规定了词法单元、语句的重组规则 → 语句的类别.

Example of C expressions 和 statements 的递归定义

- 变量名和常量是表达式 (归纳基础).
- if `expr1` and `expr2` are expressions, then:
  `expr1 '+' expr2`, `expr1 '-' expr2`, ... are expressions. (归纳条款)
- `';'` is a statement.
- if `expr` is an expression, then `expr ';'` is a stmt.
- if `stmt` is a statement and `expr` is an expr, then:
  `'if' '(' expr ')' stmt` is a stmt.

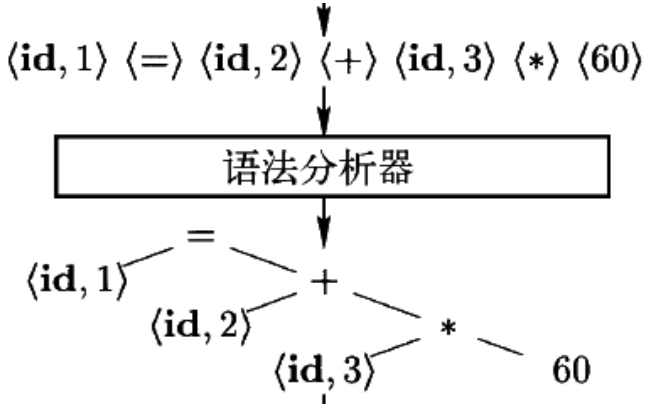
$$expr \rightarrow expr + term \mid expr - term \mid term$$
$$term \rightarrow term * factor \mid term / factor \mid factor$$
$$factor \rightarrow \mathbf{digit} \mid (expr)$$

Example of C statements

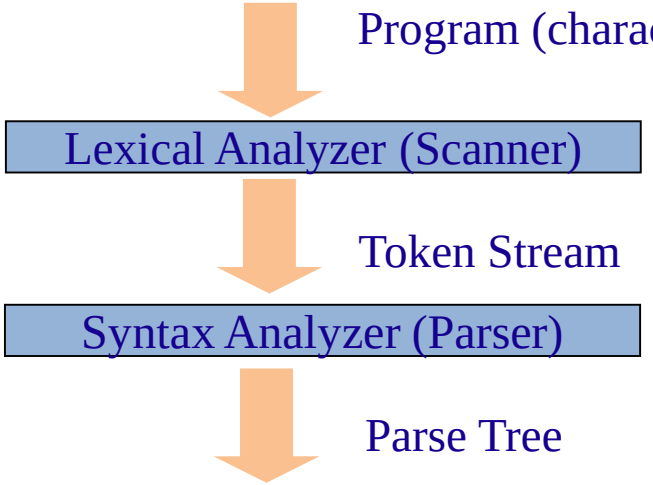
- `i = 0; while ( i < 10 ); {s = s + i ; i++;}`
- `printf ("%c\n", 3["ABCD"]);`

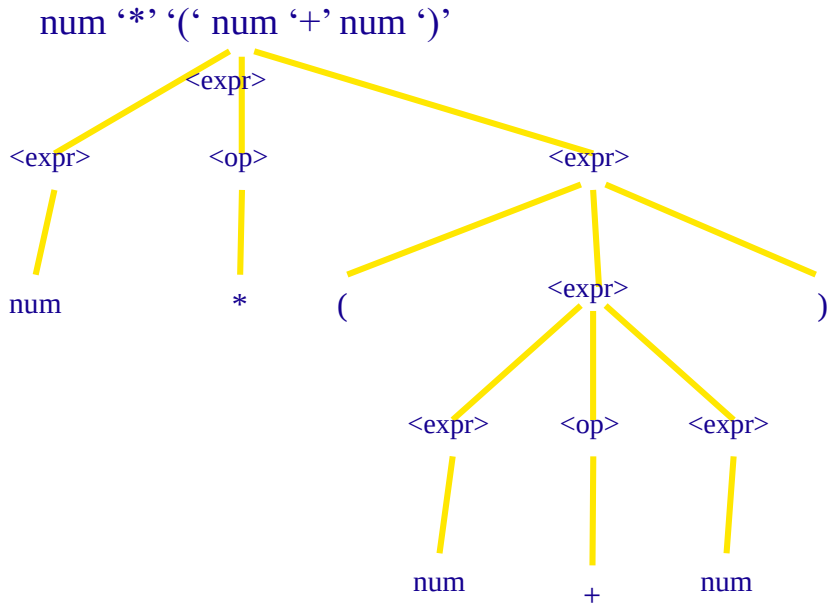
1.3 编译器组成 --- 语法分析

- **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形中间表示形式。通常是语法树（syntax tree/parse tree）
  - 指出了词法单元流的语法结构

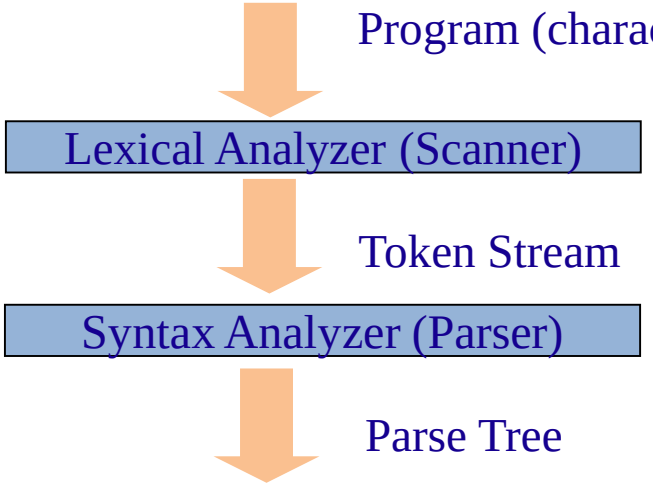


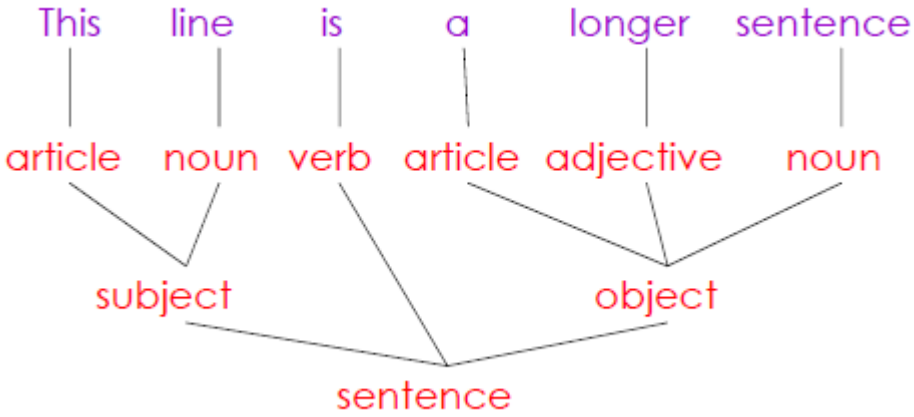
1.3 编译器组成 --- 语法分析



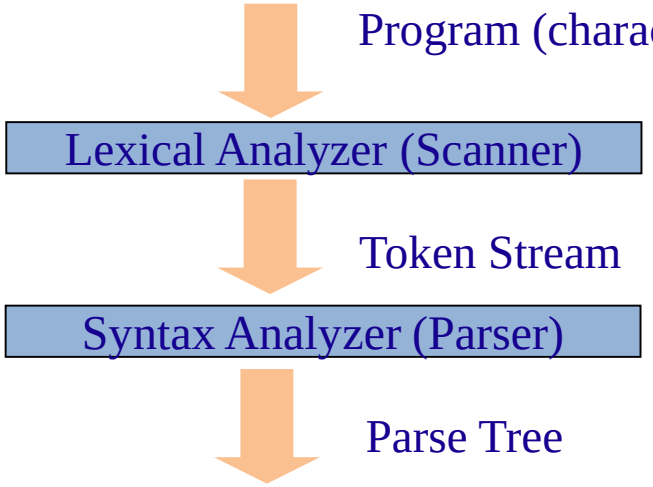


1.3 编译器组成 --- 语法分析





1.3 编译器组成 --- 语法分析




```
int * foo(i, j, k))
    int i;
    int j;
{
    for(i=0; i j) {
    fi(i>j)
        return j;
}
```

Extra parentheses

Missing increment

Not an expression

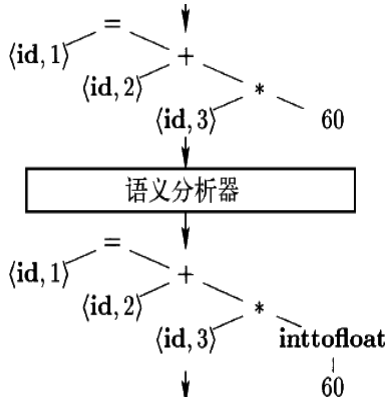
Not a keyword

1.3 编译器组成 --- 语义分析

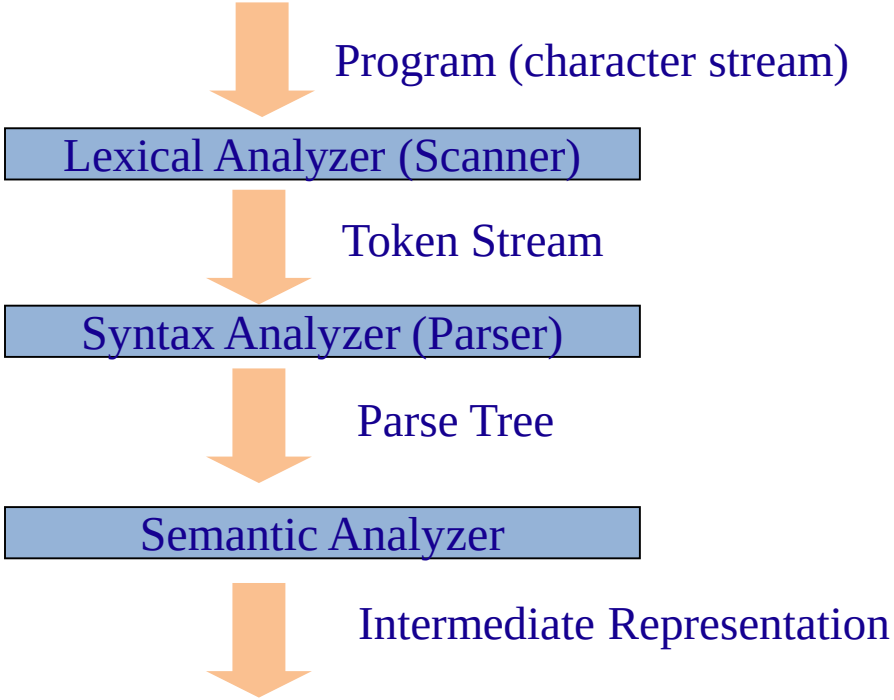
- 得到语义(meaning)，对于编译器来说比较难
- 语义分析（semantic analysis）
  - semantic: the meaning of the language.
  - 使用语法树和符号表中的信息，检查源程序是否满足语言定义的语义约束。
  - 同时收集类型信息，用于代码生成。
  - 类型检查，类型转换。

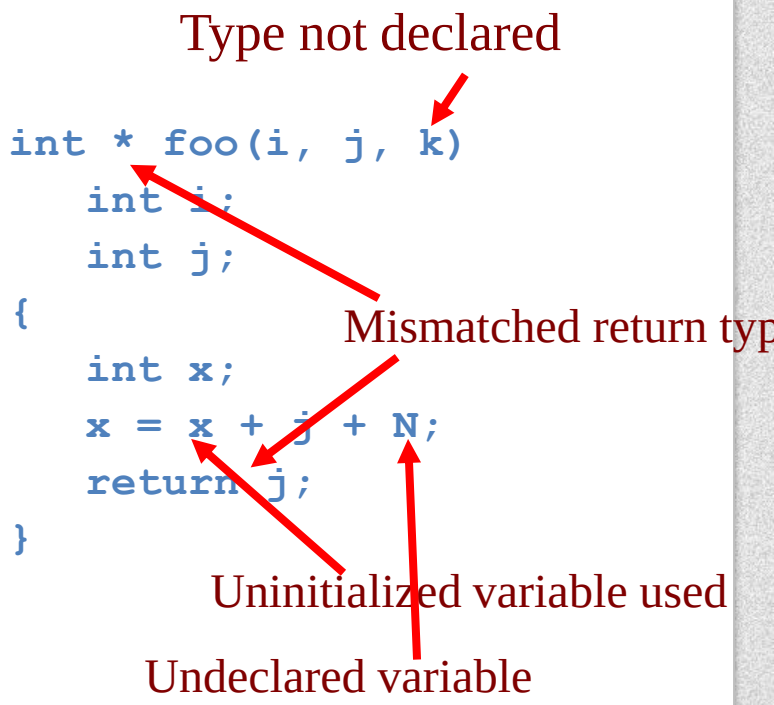
Example of C expressions

- type, value, l-value and side-effect.
- `expr1 '+' expr2` 要求两表达式的类型是可求和类型，并且一致（可能会有 implicite conversion 发生）.
- `expr1 '=' expr2` 要求表达式 1 有左值.



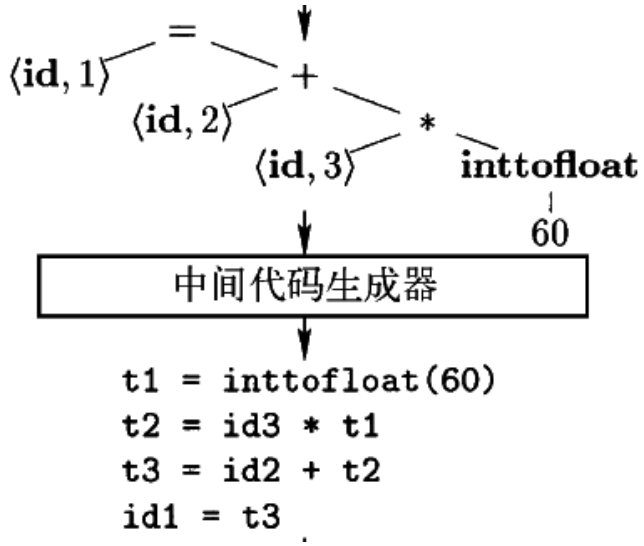
1.3 编译器组成 --- 语义分析





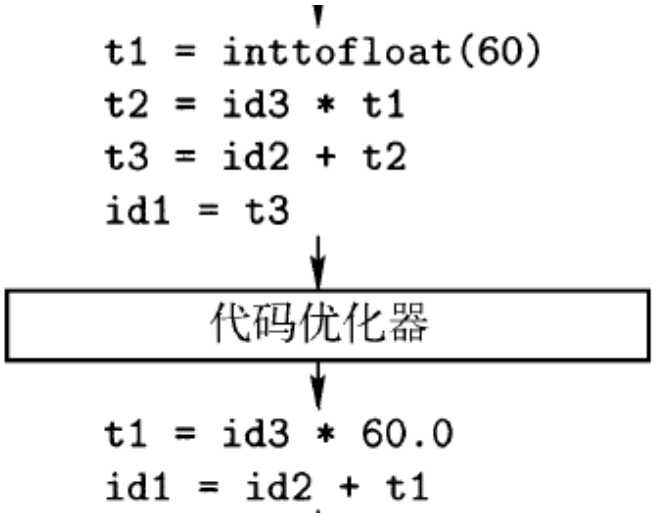
1.3 编译器组成 --- 中间代码生成

- 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

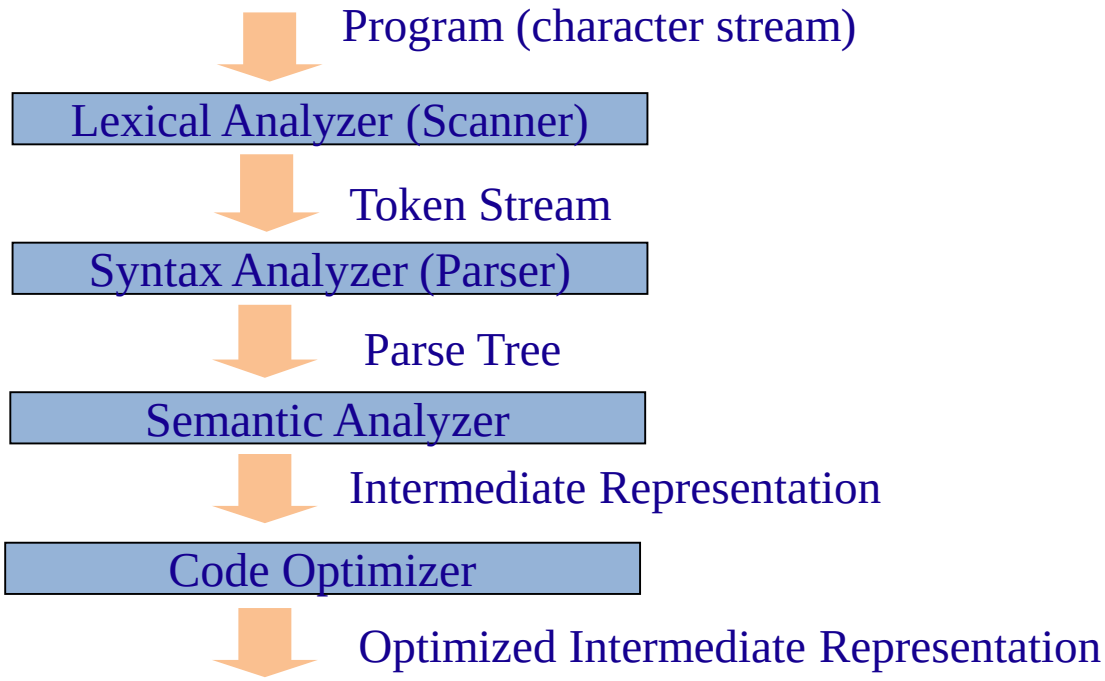


1.3 编译器组成 --- 代码优化

- 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

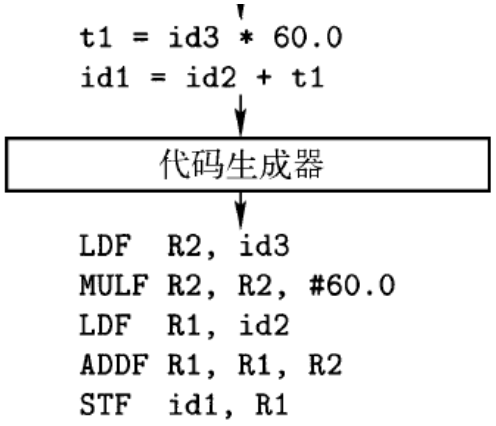


1.3 编译器组成 --- 中间代码生成

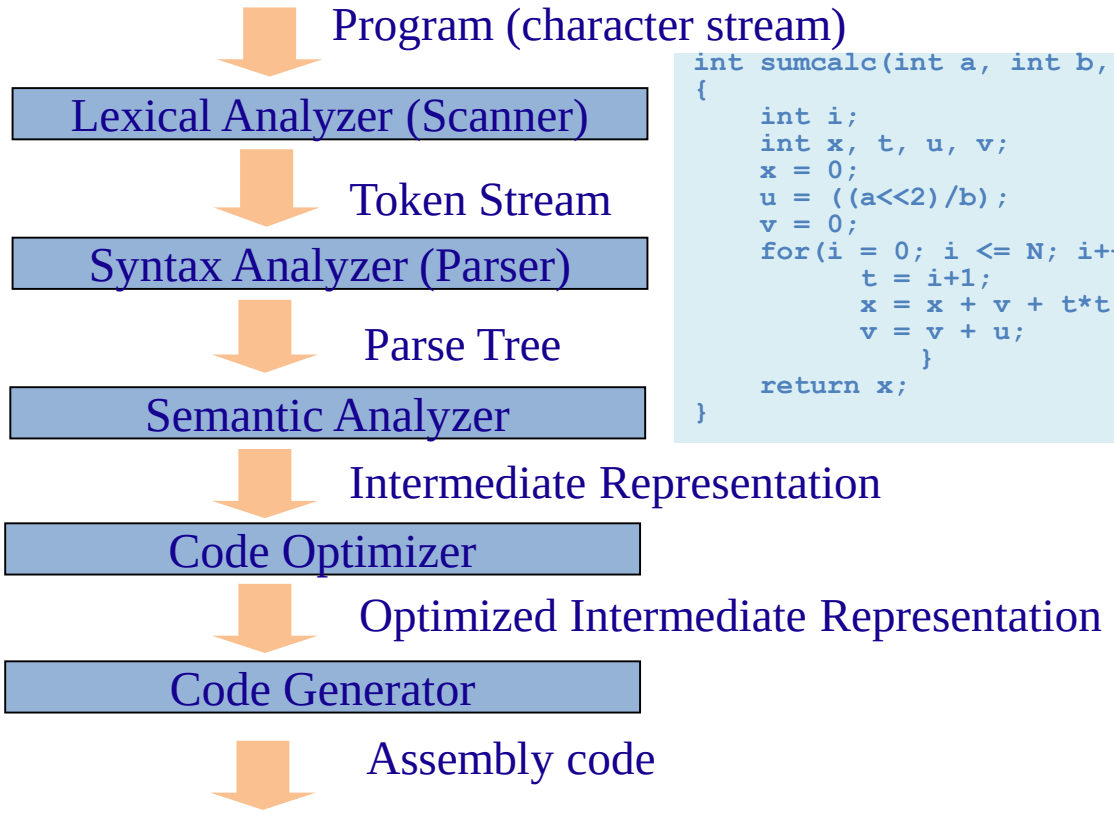


1.3 编译器组成 --- 代码生成

- **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



1.3 编译器组成 --- 代码生成




```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t, u, v;
    x = 0;
    u = ((a<<2)/b);
    v = 0;
    for(i = 0; i <= N; i++) {
          t = i+1;
          x = x + v + t*t;
          v = v + u;
            }
    return x;
}
```

```
sumcalc:
          xorl    %r8d, %r8d
          xorl    %ecx, %ecx
          movl    %edx, %r9d
          cmpl    %edx, %r8d
          jg      .L7
          sall    $2, %edi
.L5:      movl    %edi, %eax
          cltd
          idivl   %esi
          leal    1(%rcx), %edx
          movl    %eax, %r10d
          imull   %ecx, %r10d
          movl    %edx, %ecx
          imull   %edx, %ecx
          leal    (%r10,%rcx), %eax
          movl    %edx, %ecx
          addl    %eax, %r8d
          cmpl    %r9d, %edx
          jle     .L5
.L7:      movl    %r8d, %eax
          ret
```

1.3 编译器组成 --- 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

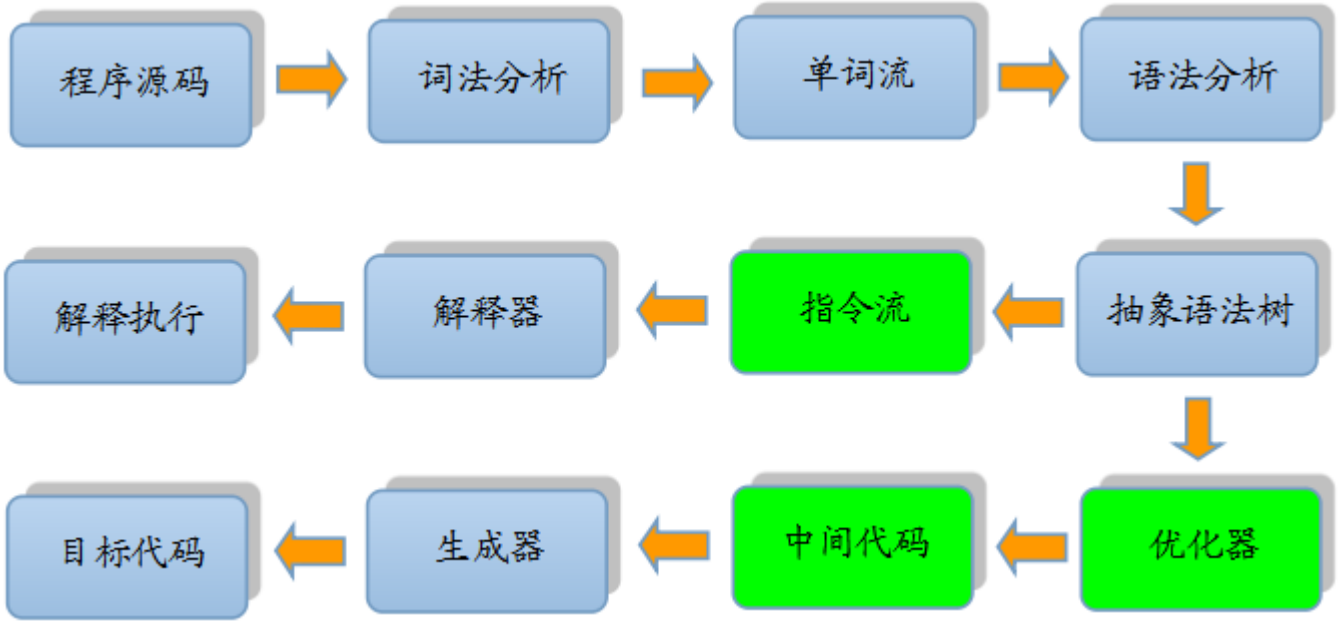
1.3 编译器组成

- 解释器(Interpreter)：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。

- 解释过程中若发现错误，则返回修改源程序，修改后重新解释执行。

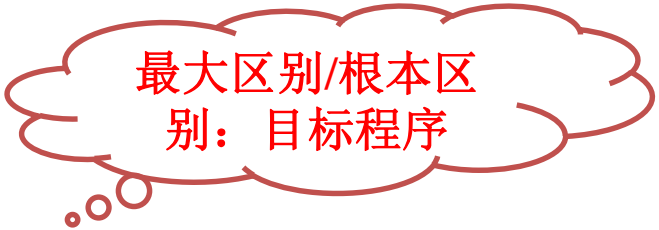
1.3 编译器组成

- **编译器和解释器的比较**



1.3 编译器组成

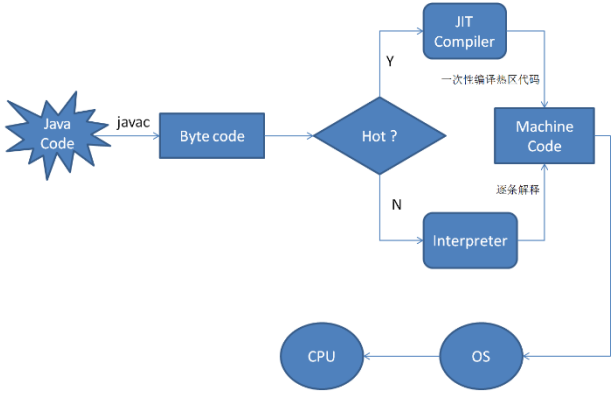
- **编译器和解释器的比较**
  - 相同点
    - 使用相同的实现技术
  - 区别
    - 实现机制: 翻译 (程序 to 程序)vs. 解释(指令 to 指令序列)



- **解释器相对于编译器的优势**
  - 可移植性好
  - 支持交互式程序设计
  - 边解释，边执行，错误诊断效果好。
- **编译器的优势在于：**
  - 效率高，一次编译，多次运行，存储代价小
  - 目标程序的执行速度比解释器快很多

1.3 编译器组成

- **Java结合了两者：**
  - javac 前端编译器：先编译成字节码(bytecode, .class文件)
  - 由JVM解释执行，可移植性好
  - JIT**即时编译器**（just-in-time compiling）



为了提高热点代码的执行效率，在运行时虚拟机将会把这些代码编译成与本地平台相关的机器码，并进行各层次的优化

- 基于采样的热点探测
- 基于计数器的热点探测

HotSpot VM: Oracle Java SE

1.3 编译器组成

- **Make**
  - 代码变成可执行文件，叫做编译（compile）
  - 安排编译的顺序，叫做构建（build）。
  - Make是最常用的构建工具，诞生于1977年，主要用于C语言的项目
    - 实际上任何只要某个文件有变化，就要重新构建的项目，都可以用Make构建。
  - 构建规则都写在Makefile, “make [选项][参数]”
- **Apache Ant --- 基于Java的构建(Build)工具**

1.4 编译器的实现

1.4 编译器实现

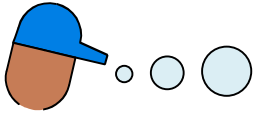
- **Efficient execution**

General

Sergeant

Foot Soldier

Cross the river and take defensive positions



Where to cross the river? Use the bridge upstream or surprise the enemy by crossing downstream?
How do I minimize the casualties??

1.4 编译器实现

- **Efficient execution**

President

My poll ratings are low,
lets invade a small nation

General

Russia or Bermuda?
Or just stall for his poll
numbers to go up?

1.4 编译器实现

- **编译程序要求**
  - Correct
    - The actions requested by the program has to be faithfully executed
  - Efficient
    - Intelligently and efficiently use the available resources to carry out the requests
    - (the word optimization is used loosely in the compiler community – Optimizing compilers are never optimal)

1.4 编译器实现

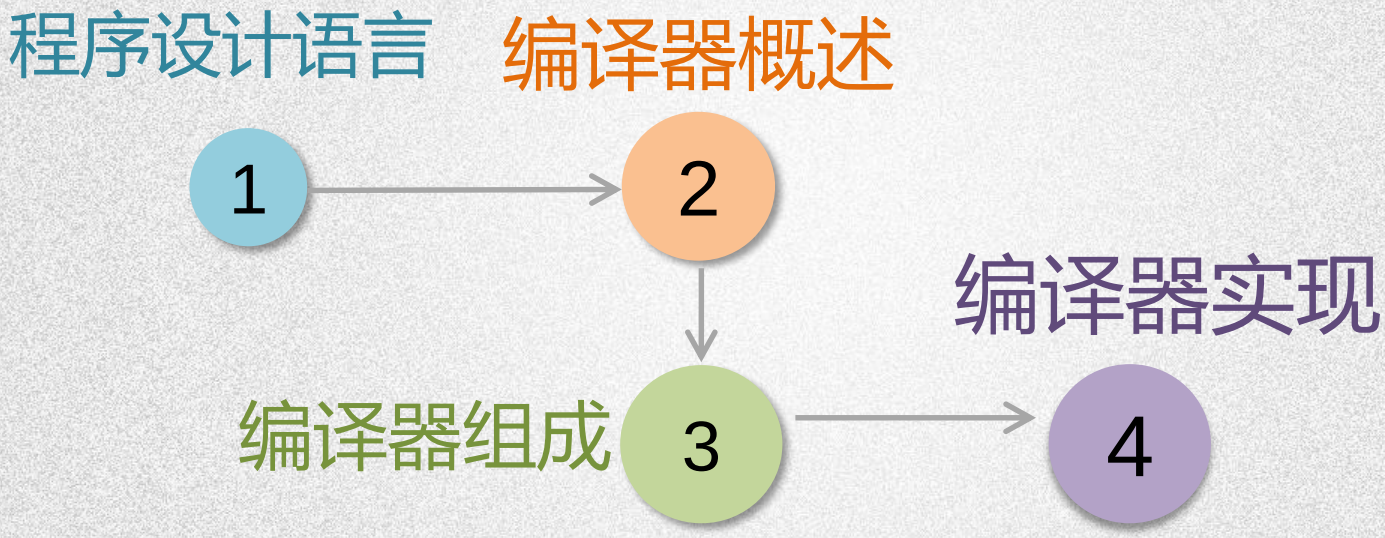
实现方法

- 编译器的开发代价是非常昂贵的，在可能的情况下，可以将一种语言的程序转换成另一种语言的程序，利用另一种语言的编译器进行编译
  - 前提条件：两种语言在语法和语义上很近似，或者一种语言是另一种语言的扩展
  - 实例：C++ → C

1.4 编译器实现

- **构造工具**
  - 扫描器生成器
  - 语法分析器生成器
  - 语法制导的翻译引擎
  - 代码生成器的生成器
  - 数据流分析引擎
  - 编译器构造工具集

本章小结



课后作业

- 教材Page 2: 1.1.3, 1.1.4
- 列举一些知名的编译器构造工具，并说明他们分别的作用

Thank you!