COMP130014 编译

第一讲：课程介绍

徐 辉

xuh@fudan.edu.cn

主要内容

一、课程介绍

二、编译：以计算器为例

三、编译流程概览

一、课程介绍

教学团队

- 授课教师：徐辉
  - Ph.D, CUHK
  - 研究方向：程序分析、软件可靠性
  - 主页：https://hxuhack.github.io

江湾校区 交叉二号楼D6023
xuh@fudan.edu.cn

- 助教：

陈实力（Head TA）
slchen24@m.fudan.edu.cn

王兆瀚
zhaohanwang24@m.fudan.edu.cn

柏露
24210240001@m.fudan.edu.cn

董方
fdong22@m.fudan.edu.cn

张涵星
zhanghx@m.fuden.edu.cn

陈可
kchen21@m.fudan.edu.cn

课程信息

- 课堂教学：
  - 时间：星期四 6-8节（1:30pm-4:10pm）[1-16周]
  - 地点：光华楼西207室（HGX207）
- 上机实践：
  - 时间：星期四9-10节（4:20am-6:00pm）[1-16周]
  - 地点：H逸夫楼302、402
- 课程平台：
  - 官方平台：Elearning
  - 课程主页：https://github.com/hxuhack/course_compiler
  - 讨论：WeChat

为什么学习编译原理？

- 编译器是程序员和计算机沟通的桥梁
- 通过便于理解的高级语言提升软件开发效率

源代码

```
int main(){
  printf("hello,
     compiler!\n");
  return 0;
}
```

汇编

```
push    rax
mov     edi, offset s
call    _puts
xor     eax, eax
pop     rcx
retn
```

机器码

```
50 BF 04 20
40 00 E8 F5
FE FF FF 31
C0 59 C3 90
```

第一门被广泛使用的通用高级编程语言是？

Fortran：1954年

- 23500行汇编代码，耗费人力18人年
- 很多先进思想至今沿用，以操作符优先级解析为例
  - 将“+”或“-”替换为“))+((”或“))-((”
  - 将“*”或“/”替换为“)*(”或“)/(”
  - 在程序开头添加“((”，结尾添加“))”

解析

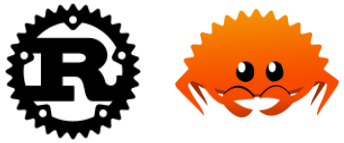
```
A + B * C   ⇒   ((A)) + ((B) * (C)))
```

优化

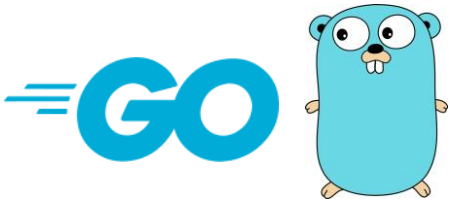
```
V4 = B                          V2 = B * C
V5 = C                          A + V2
V2 = V4 * V5   ⇒
V3 = A
V1 = V3
V1 + V2
```

David Padua. "The fortran I compiler." *Computing in Science & Engineering* 2, no. 1 (2000): 70-75.

新需求=>新型编程语言和编译技术不断出现



Mozilla（浏览器引擎）
Graydon Hoare
2006-2014（v1）



Google（多核、分布式服务）
R. Griesemer, R. Pike, K. Thompson
2007-2012（v1）

Apple（应用程序）
Chris Lattner
2010-2014（v1）

Modular（人工智能）
Chris Lattner
2022-2024（v?）

TIOBE语言使用统计排名

Aug 2024	Aug 2023	Programming Language	Ratings	Change
1	1	Python	18.04%	+4.71%
2	3	C++	10.04%	-0.59%
3	2	C	9.17%	-2.24%
4	4	Java	9.16%	-1.16%
5	5	C#	6.39%	-0.65%
6	6	JavaScript	3.91%	+0.62%
7	8	SQL	2.21%	+0.68%
8	7	Visual Basic	2.18%	-0.45%
9	12	Go	2.03%	+0.87%
10	14	Fortran	1.79%	+0.75%
11	13	MATLAB	1.72%	+0.67%
12	23	Delphi/Object Pascal	1.63%	+0.83%
13	10	PHP	1.46%	+0.19%
14	19	Rust	1.28%	+0.39%
15	17	Ruby	1.28%	+0.37%
16	18	Swift	1.28%	+0.37%
17	9	Assembly language	1.21%	-0.13%
18	27	Kotlin	1.13%	+0.44%
19	16	R	1.11%	+0.19%
20	11	Scratch	1.09%	-0.13%

https://www.tiobe.com/tiobe-index/

编译器和编程语言的重要性

- 计算机领域最高荣誉：ACM Turing Award

2020年
Aho & Ullman

Subtype

2008年
Barbara Liskov

optimization

2006年
Frances Allen

BNF范式, ALGOL 60

2005年
Peter Naur

Hoare Logic

1980年
Tony Hoare

C, Unix

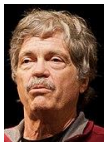
1983年
Dennis Ritchie

Pascal

1984年
Niklaus Wirth

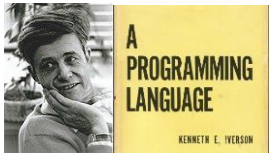
OOP: Simula

2001
Dahl & Nygaard

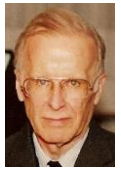


OOP: Smalltalk

2003年
Alan Kay



1979年
Kenneth Iverson



Fortran

1977年
John Backus

Finite Automata

1976年
Dana Scott

ALGOL 60

1972年
Edsger Dijkstra

Lisp, AI

1971年
John McCarthy

ALGOL

1966年
Alan Perlis

first
successful
compiler

自然语言可以编程吗？

抽象级别

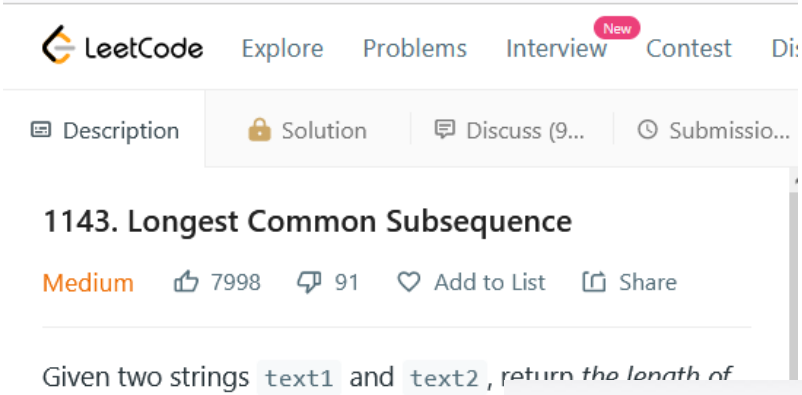
自然语言

伪代码编程

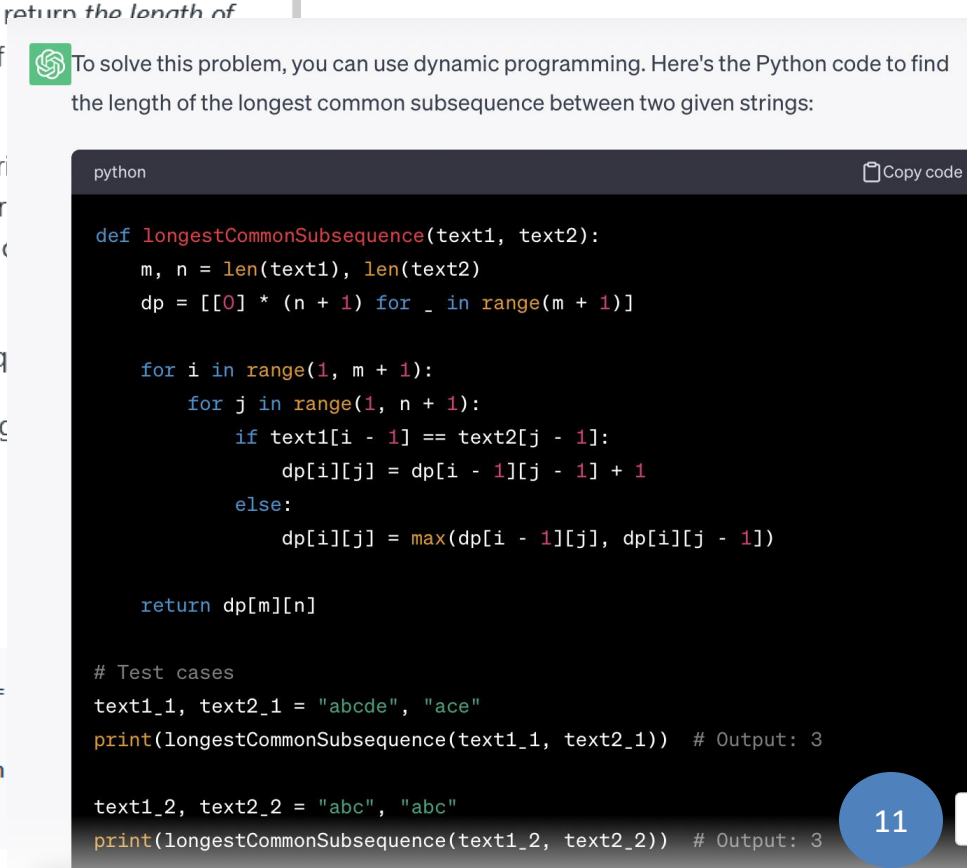
高级编程语言

汇编语言

机器码



LLM好像可以？
但正确性难以保证



知乎上的讨论

如何学习编译原理?

https://www.zhihu.com/question/21515496

知乎用户

编程话题下的优秀答主

434 人赞同了该回答

如何学习编译原理？个人不太建议一上手就拿起龙书、虎书等等来看。

学过编译原理课程的同学应该有体会，各种文法、各种词法语法分析算法，非常消磨人的耐心和兴致；中间代码生成和优化，其实在很多应用场景下并不重要（当然这一块对于"编译原理"很重要）；语义分析要处理很多很多细节，特别对于比较复杂的语言；最后的指令生成，可能需要读各种手册，也比较枯燥。

CompilerCoder

GPU编译器工程师

138 人赞同了该回答

大学的时候学过一门编译原理的课程，当时老师讲课主要讲的是词法分析、语法分析等，对于后端基本没讲。当时讲各种文法的时候一上来就是各种符号，各种概念非常绕，最后为了考试只能硬学。

ddss

79 人赞同了该回答

這是個好問題, 我光是發現怎麼學習編譯原理就花了不少時間, 也買了不少書, 但每本書的實作都不同, 讓學習更難了。

最後我想到一個方法:

我要實作 c 語言編譯器, 畢竟書上寫的 pascal 實作我一點都不感興趣, 我又沒在用 pascal, 我在使用的是 c/c++ 語言, 實作一個自己沒在用的語言實在是沒有動力。

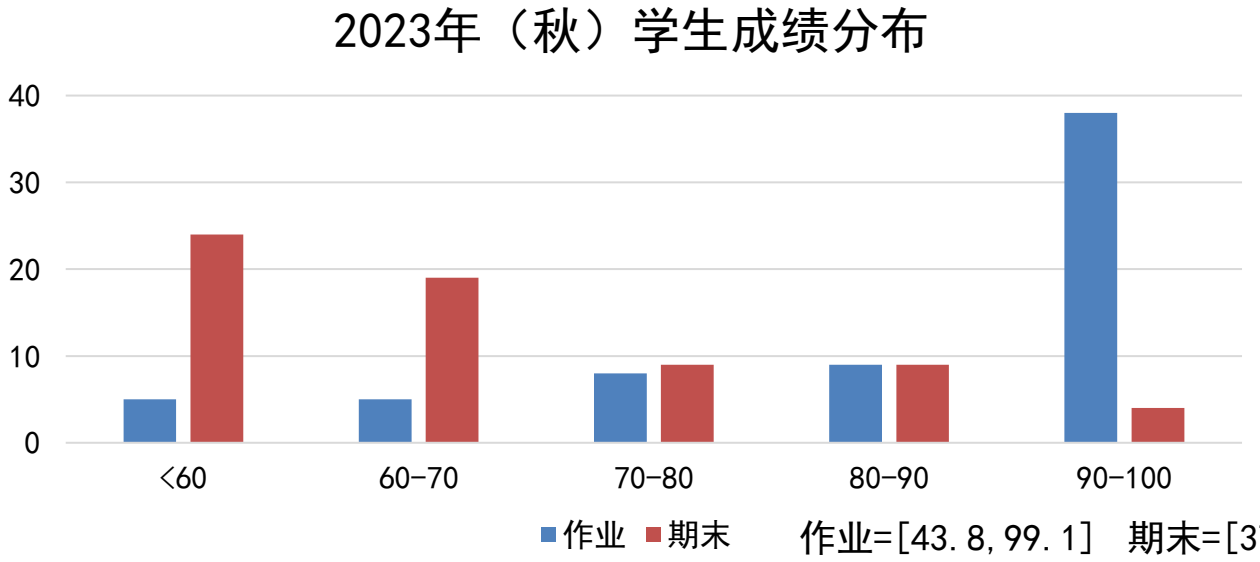
教学大纲设计（Tentative）

周	时间	授课内容	上机内容		
1	Sep-5	课程入门	通过LLVM了解编译器		
2	Sep-12	词法分析	Flex/Bison	布置HW1	
3	Sep-19	上下文无关文法	答疑		
4	Sep-26	自顶向下分析	答疑		
5	Oct-3	假期			
6	Oct-10	自底向上分析			验收HW1
7	Oct-17	类型推导	类型推导	布置HW2	
8	Oct-24	线性中间代码	AST => 线性IR	布置HW3	
9	Oct-31	静态单赋值形式	线性IR => SSA	布置HW4	
10	Nov-7	过程内优化			验收HW2
11	Nov-14	过程间优化			验收HW3
12	Nov-21	指令选择		布置HW5/6	
13	Nov-28	寄存器分配			验收HW4
14	Dec-5	后端优化			
15	Dec-12	AI编译器			
16	Dec-19	HW6报告			验收HW5

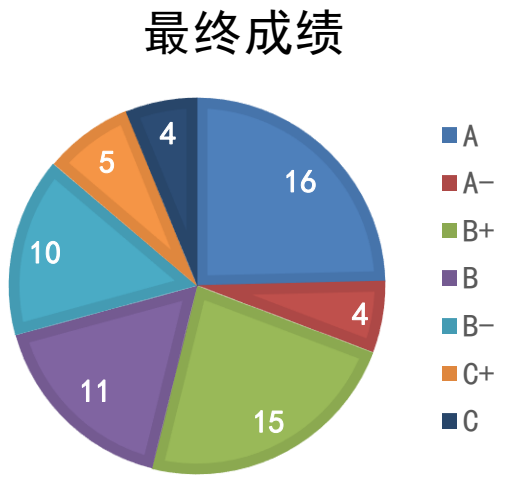
课程考核

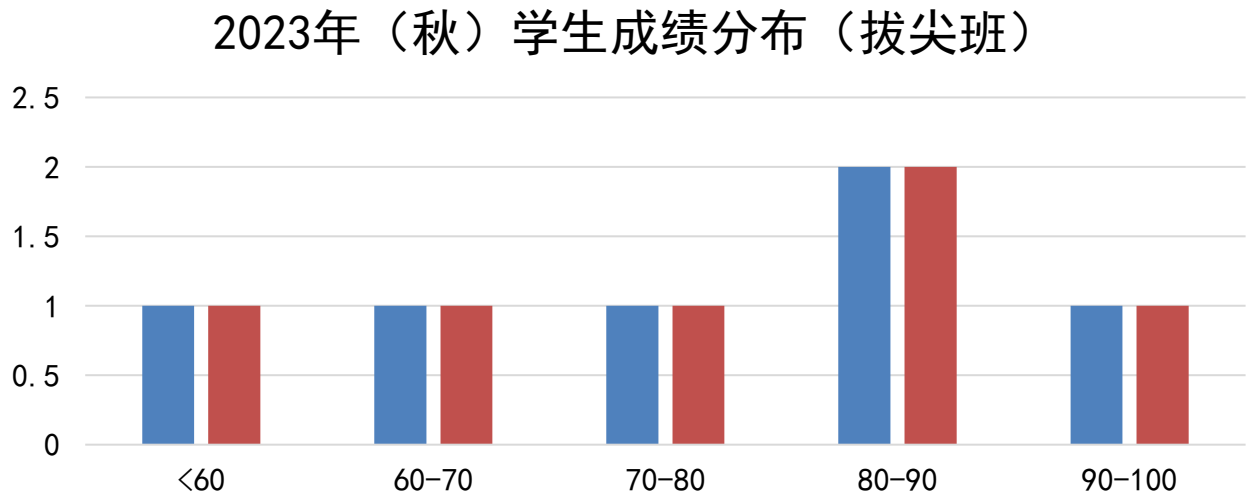
- 平时成绩：50%
  - 普通班：5次实验：各占比20%
  - 拔尖班：5次实验（同普通班）+ 1次自选实验，分数占比60% vs 40%
- 开卷考试：50%
  - 2024-12-25 15:30~17:30

往年学生成绩

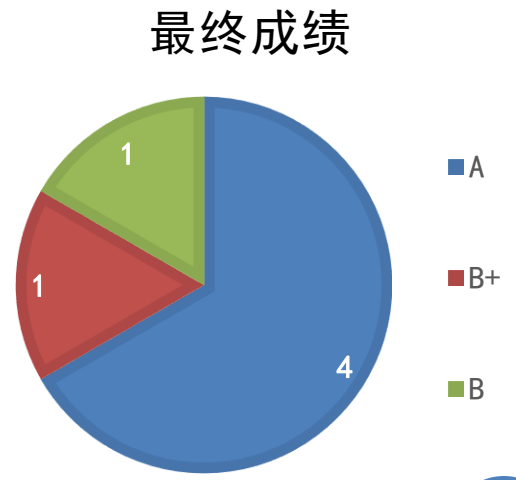


作业=[43.8, 99.1]　期末=[37.5, 96]

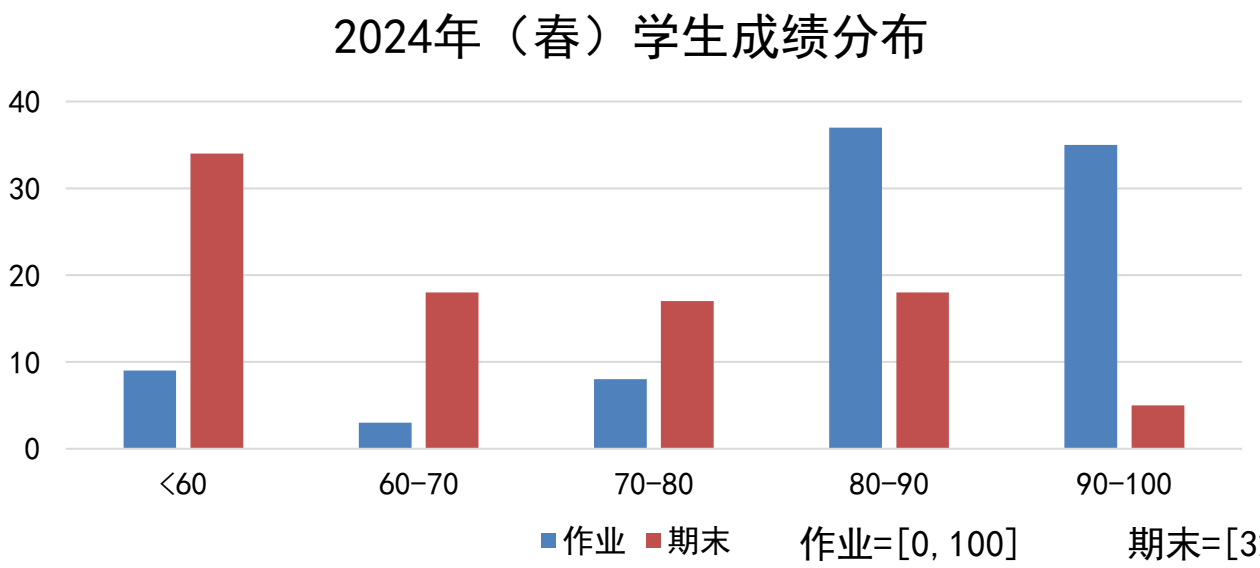


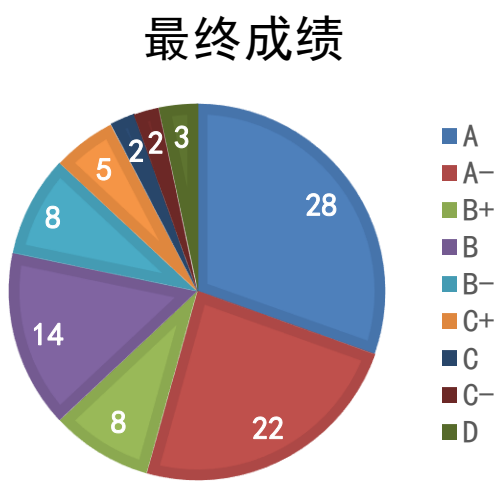


作业=[48.3, 95.9]期末=[59, 96]



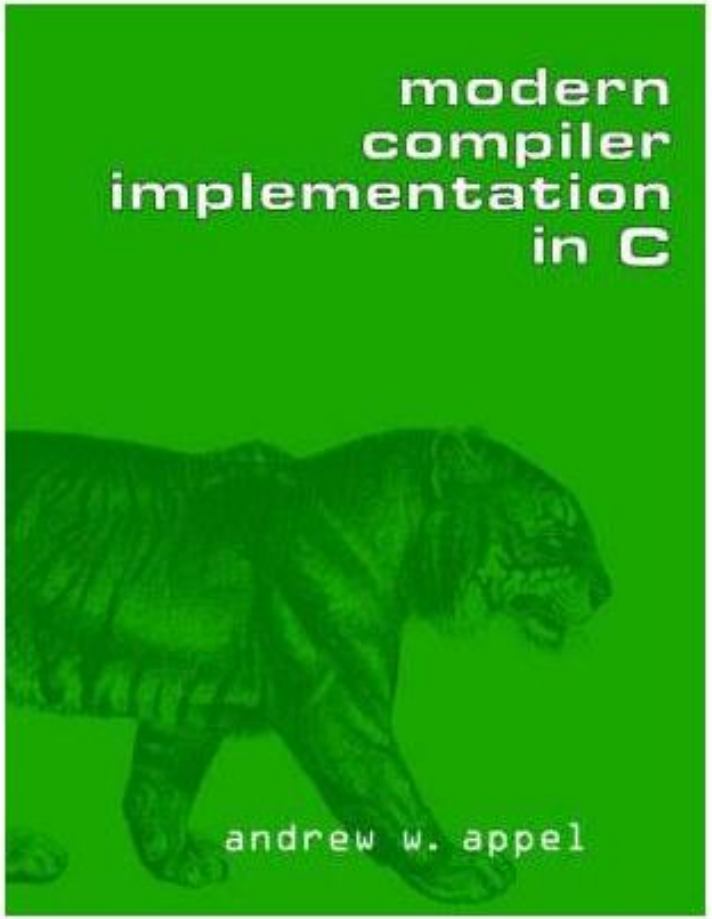
往年学生成绩

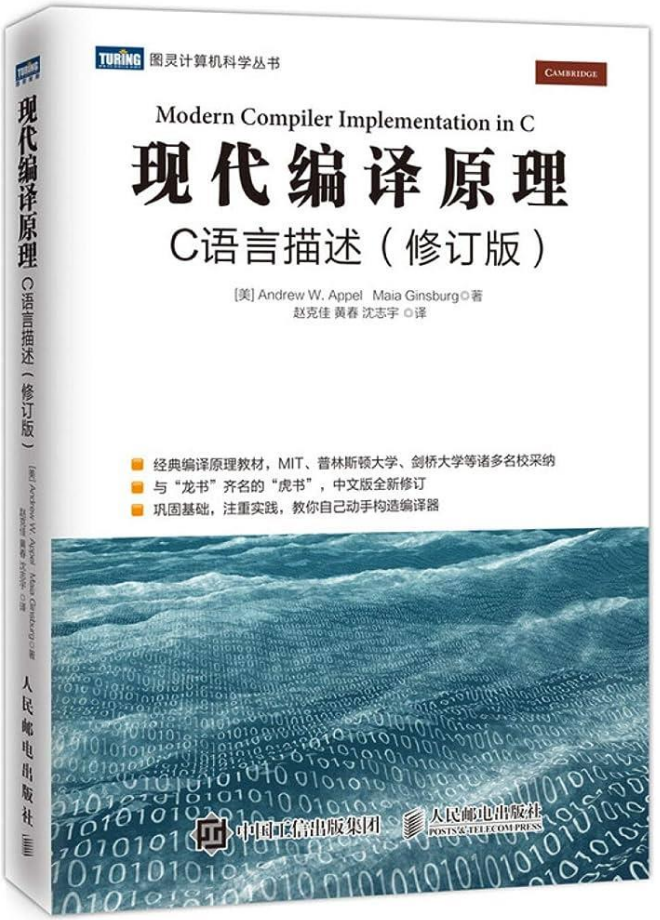

2024年（春）学生成绩分布
40
30
20
10
0
<60
60-70
70-80
80-90
90-100
作业
期末
作业=[0, 100]
期末=[32, 94]


最终成绩
28
22
8
14
8
5
2
2
3
A
A-
B+
B
B-
C+
C
C-
D

主要参考书

- 自编讲义为主
- 参考书：现代编译原理， Andrew W.Appel, Maia Ginsburg著

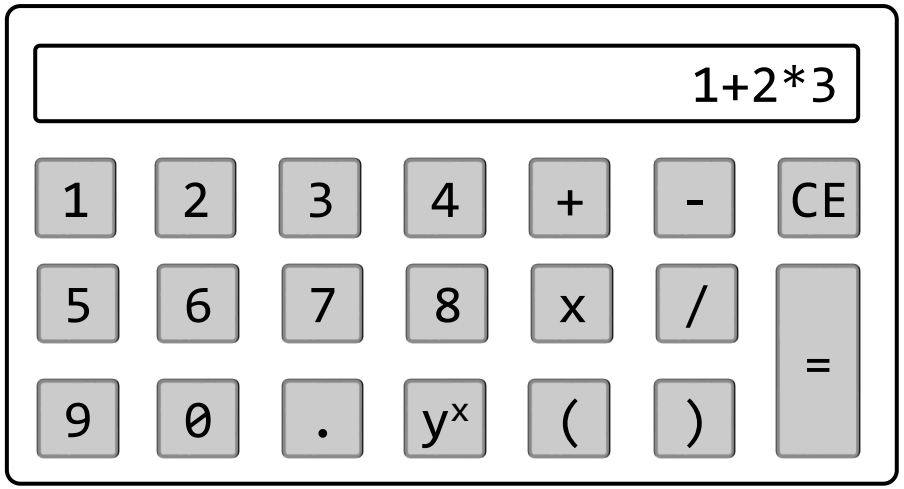




二、编译：以计算器为例

如何实现一个计算器？

- 操作数：整数、浮点数、负数，如123、0.1、-0.1
- 运算符：加、减、乘、除四则运算和指数运算
- 支持括号



步骤1：识别操作数、运算符和括号

```
Input: character stream;
Output: token stream;

while (true) {
    cur = charStream.next()
    match (cur) {
        '0-9' => ...
        '+' => ...
        '-' => ...
        '*' => ...
        '/' => ...
        '^' => ...
        '(' => ...
        ')' => ...
        _ => break;
    }
}
```

操作数需要考虑多个数字的情况

负号 or 减号？

难点：如何区分“-”是负号还是减号

```
Input: character stream;
Output: token stream; //保存解析结果

while (true) {
    cur = charStream.next()
    match (cur) {
        '0-9' => num.append(cur),
        '+' => {tok.add(num); tok.add(ADD); num.clear(); }
        '-' => { ... }
        '*' => {tok.add(num); tok.add(MUL); num.clear(); }
        '/' => {tok.add(num); tok.add(DIV); num.clear(); }
        '^' => {tok.add(num); tok.add(POW); num.clear(); }
        '(' => {tok.add(num); tok.add(LPAR); num.clear(); }
        ')' => {tok.add(num); tok.add(RPAR); num.clear(); }
        _ => break,
    };
}
```

步骤2：分析算式含义：合规性

1*2+3 ✓

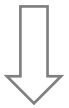
1+2*3 ✓

1+2*3^4*5 ✓

1**2+3 ✗

1+((2+3) ✗

1--2+3 ✗



1-(-2)+3 ✓

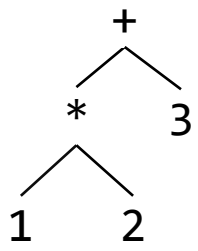
通过限制表达能力解决负号问题

步骤2：分析算式含义：优先级

- 指数运算优先级 > 乘除运算 > 加减运算

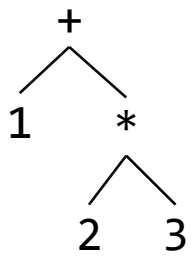
1*2+3

(1*2)+3



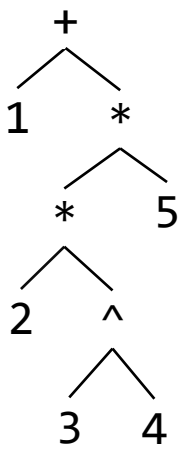
1+2*3

1+(2*3)



1+2*3^4*5

1+((2*(3^4))*5)



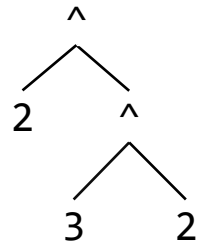
Fortran方法：(((1)))+(((2))*((3)^(4))*((5)))

步骤2：分析算式含义：结合性

- 加减乘除运算为左结合
- 指数运算为右结合

2^3^2

2^(3^2)

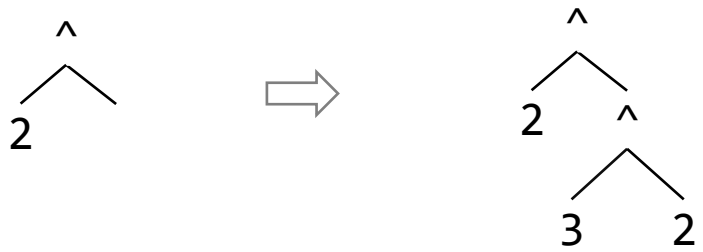


(((2)^(3)^(2)))

运算符优先级解析思路

- 使用栈记录已经遍历的运算符
- 如果遇到的运算符为^，将其作为栈顶运算符的右孩子节点

2^3^2

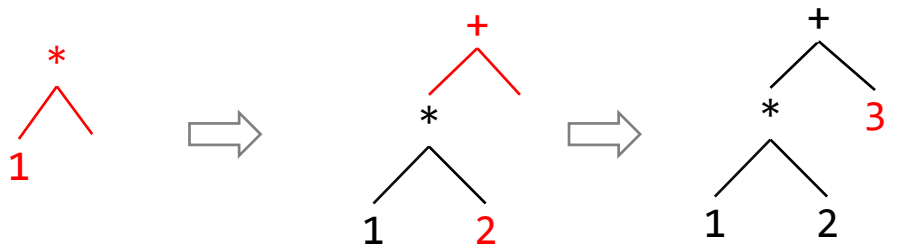


1+3^2

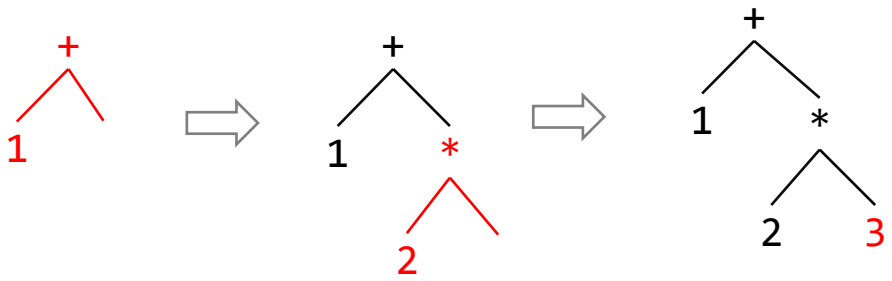
运算符优先级解析思路

- 如果当前遇到的运算符为左结合
  - 如果当前运算符≤栈顶运算符的优先级，将其作为父节点
  - 如果当前运算符>栈顶运算符的优先级，将其作为右孩子节点

1*2+3

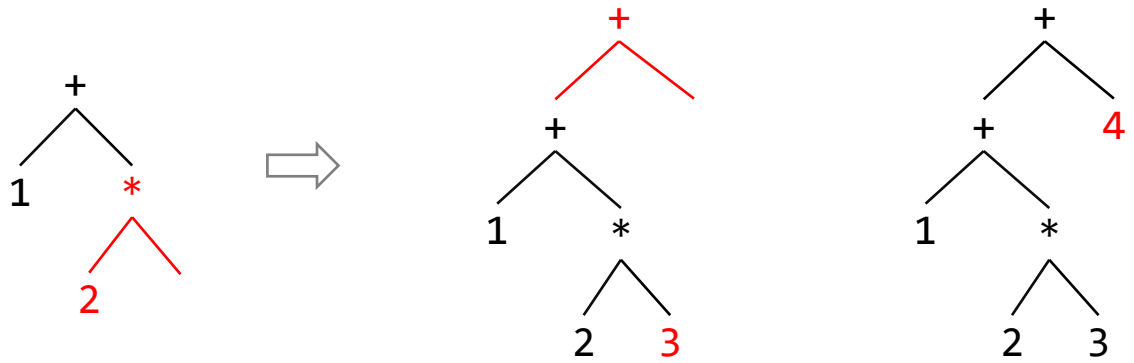


1+2*3

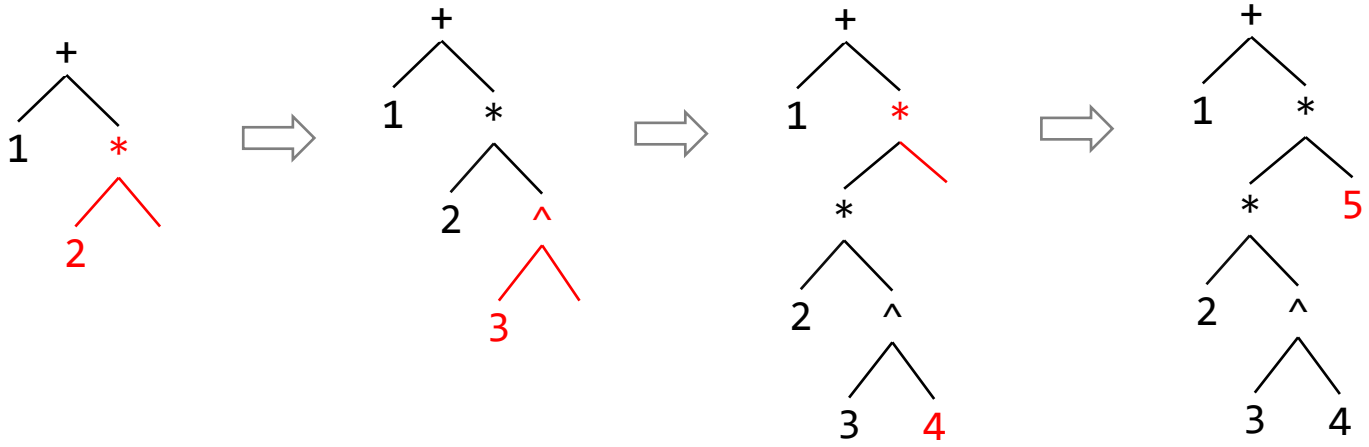


运算符优先级解析思路

1+2*3+4



1+2*3^4*5

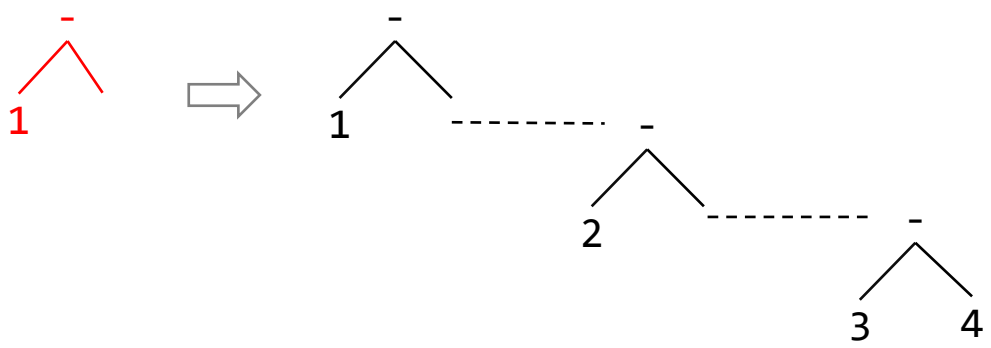


如果当前运算符≤栈顶运算符的优先级：

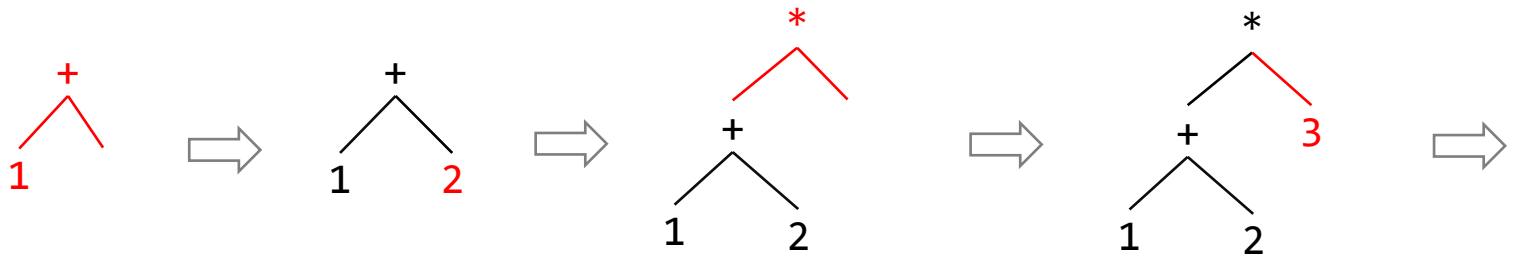
let top = s.pop() until current op > top

运算符优先级解析思路

(1-(2-(3-4)))　　遇到“（”，递归



((1+2)*3)^(4*5)

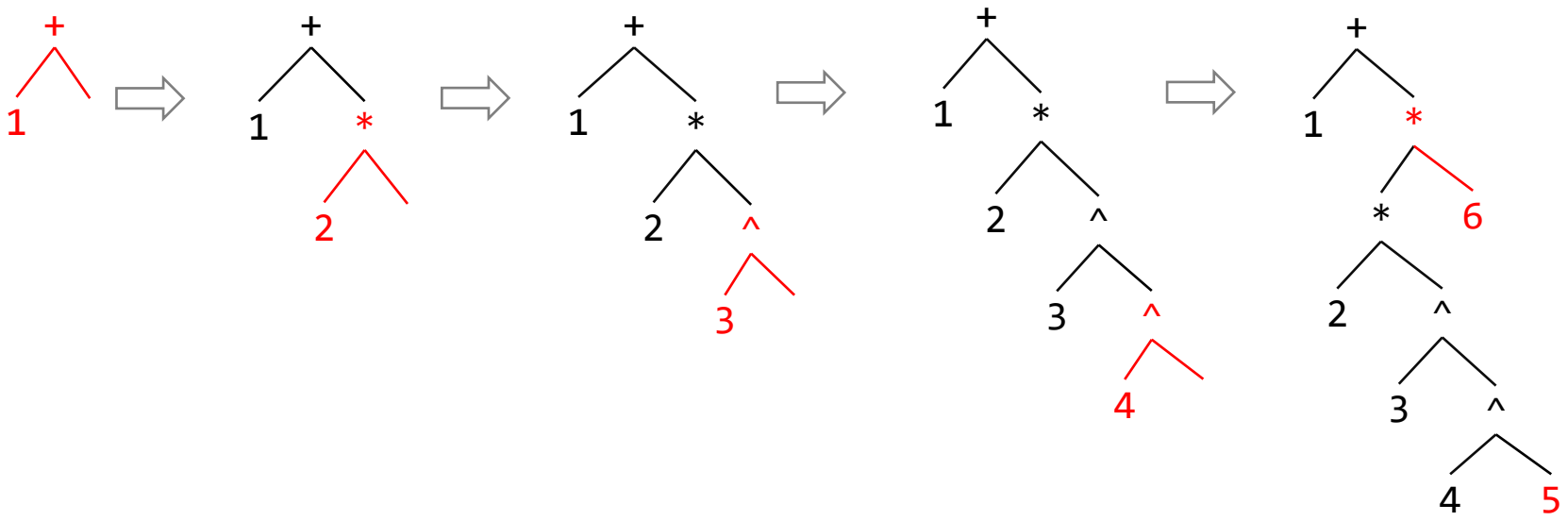


使用优先级标记

```
Pred[ADD] = 1,2
Pred[SUB] = 1,2
Pred[MUL] = 3,4
Pred[DIV] = 3,4
Pred[POW] = 6,5
```

初始化优先级 0 1 2 3 4 6 5 6 5 3 4 0

算式 1 + 2 * 3 ^ 4 ^ 5 * 6



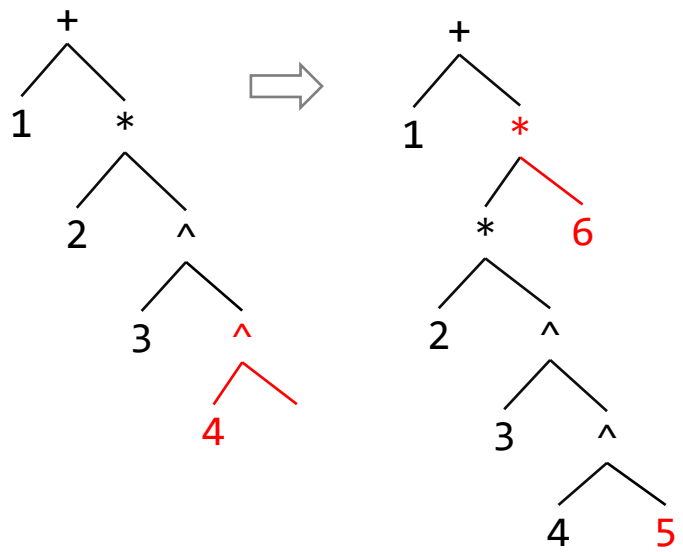
算法实现参考

```
Preced[ADD] = 1,2; Preced[SUB] = 1,2;
Preced[MUL] = 3,4; Preced[DIV] = 3,4;
Preced[POW] = 6,5; Preced[EOF] = 0,0;

Input: tokstream; // token序列
Output: left; // 二叉树
PrattParse(cur, precedence) -> BinTree {
    left = cur.next();
    if left.type != tok::NUM
        exit -1;
    while true:
        let op = cur.peek();
        if op.tokentype == tok::NUM
            exit -1;
        lp, rp = Preced[peek];
        if lp < precedence
            break;
        cur.next();
        right = PrattParse(cur, rp)
        left = CreateBinTree(op, left, right)
    return left;
}
```

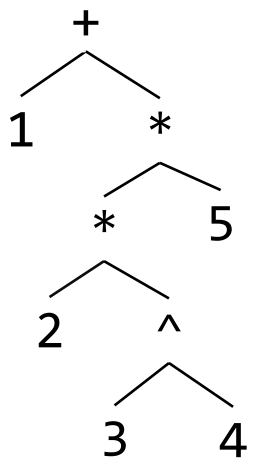
0 1 2 3 4 6 5 6 5 3 4 0

1 + 2 * 3 ^ 4 ^ 5 * 6

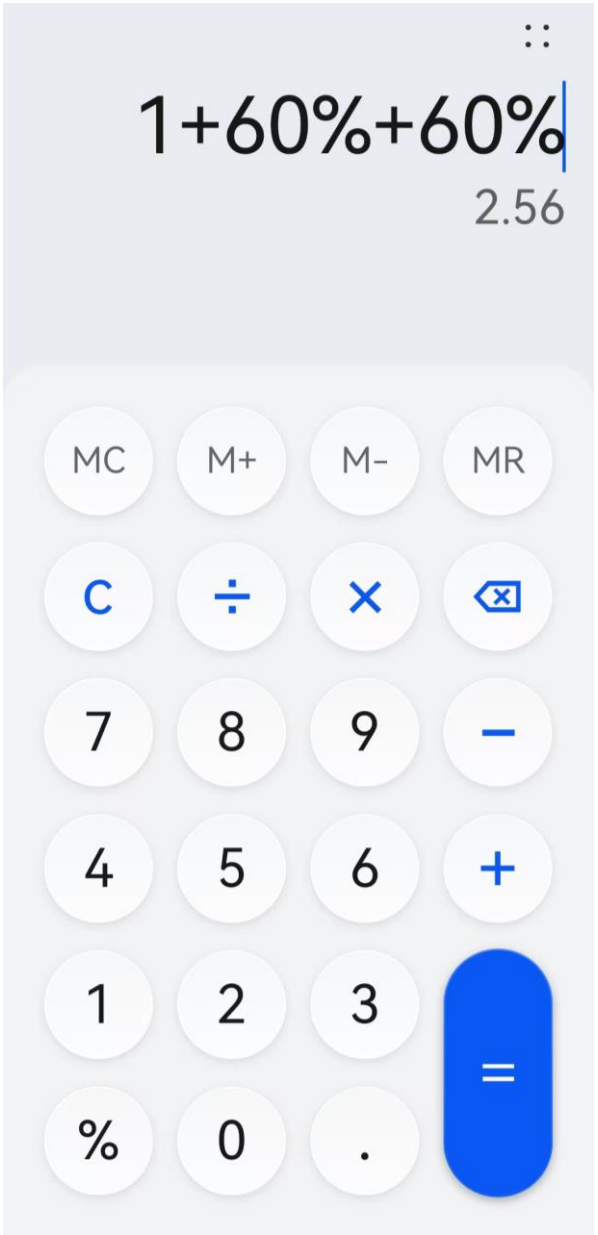


步骤3：解释执行/翻译为逆波兰表达式

- 前序遍历语法解析树=>波兰表达式
  - + 1 * * 2 ^ 3 4 5
  - 满二叉树无歧义
- 后序遍历语法解析树=>逆波兰表达式
  - 1 2 3 4 ^ * 5 * +
- 逆波兰表达式方便计算：
  - 顺序读取，遇到操作数则入栈
  - 遇到运算符，则弹出栈顶的两个操作数，求值后将结果入栈



计算器Bug?

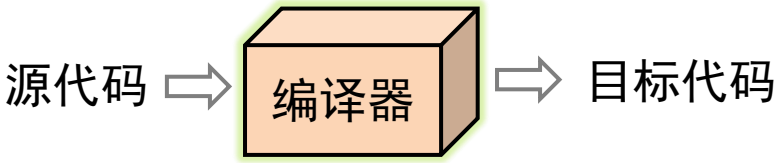


- 为什么这样?
- 如何实现的?

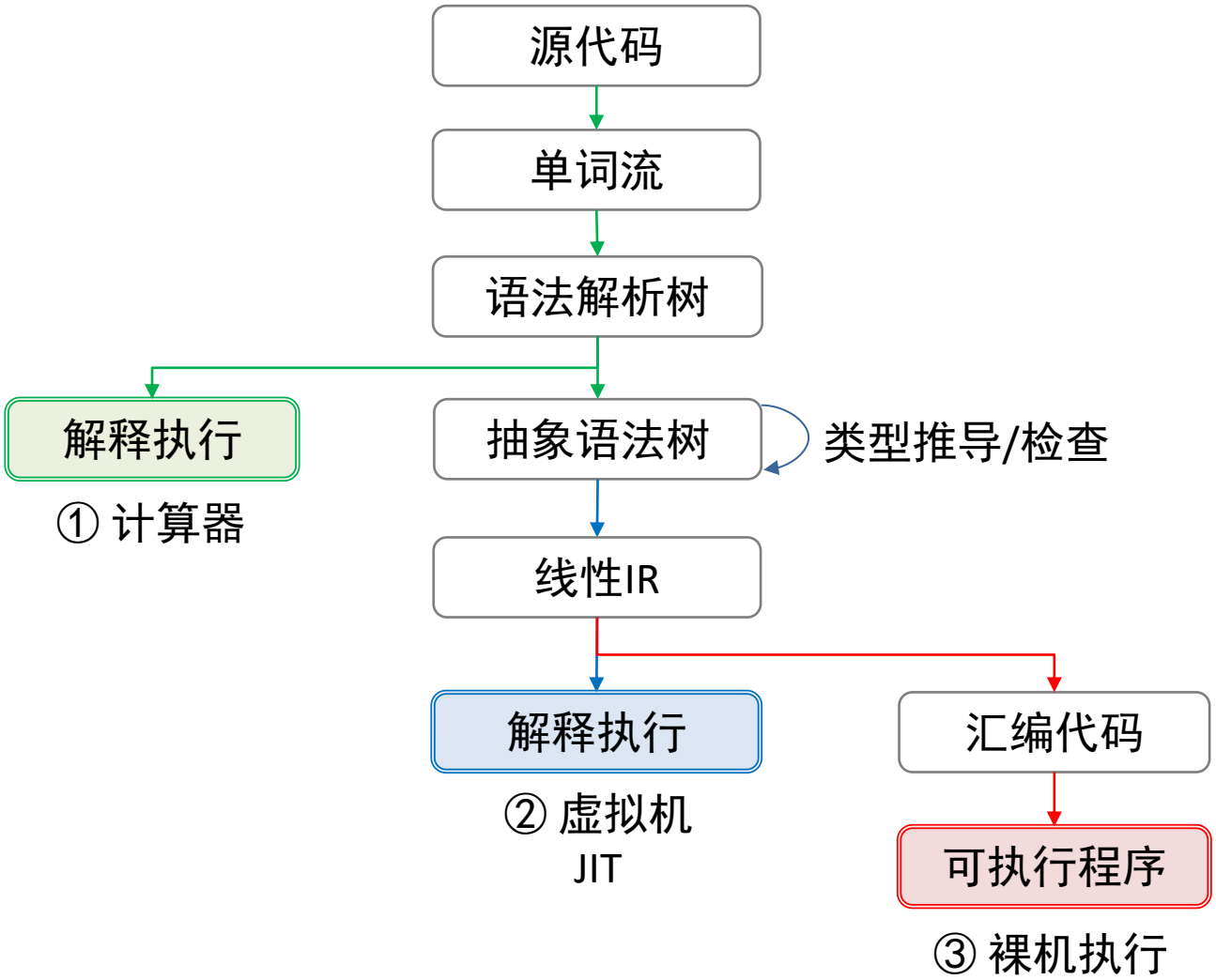
三、编译流程概览

编译

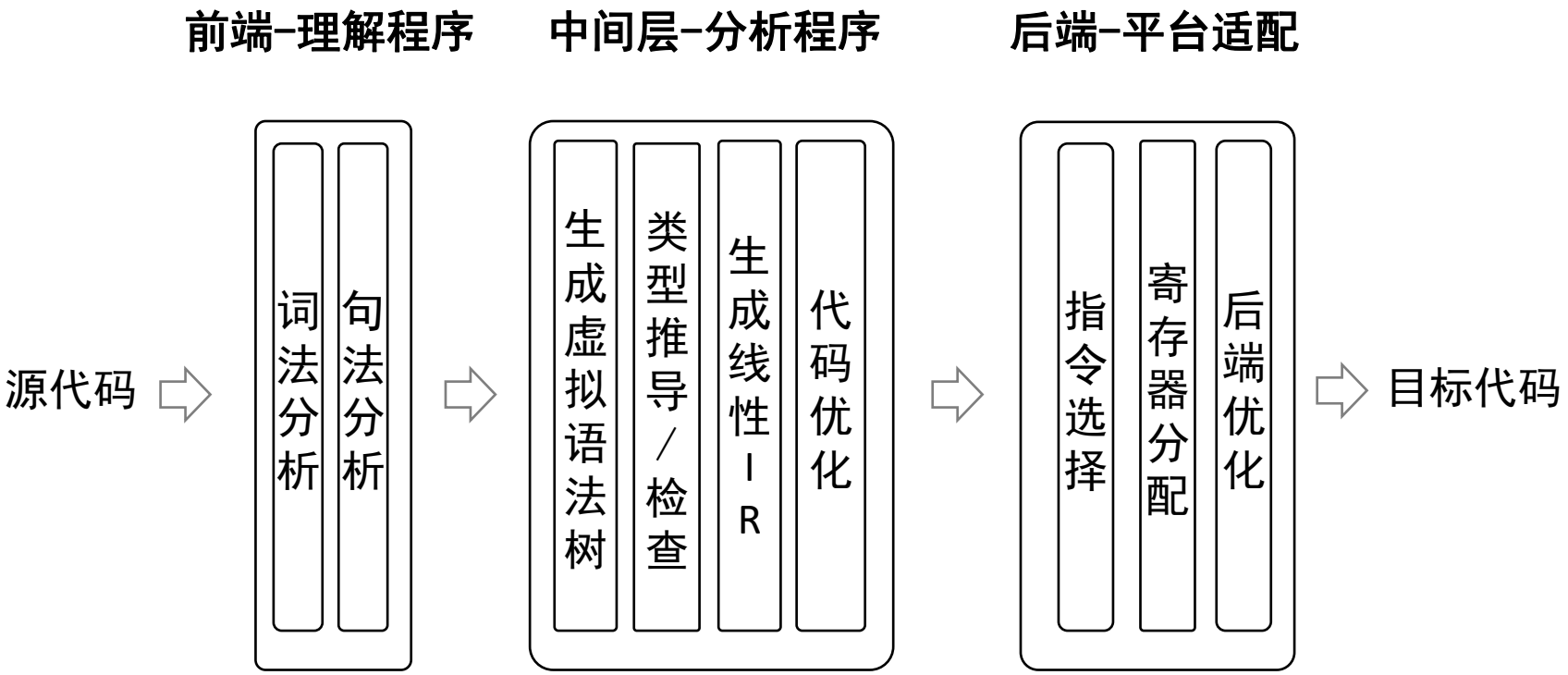
- 从一种程序语言转换为另一种程序语言
  - 源代码=>中间代码（解释执行）
    - Java/Python => Bytecode
    - C/C++/Rust => WebAssembly
  - 源代码=>汇编代码/可执行程序（编译执行）
    - C/C++/Rust => X86/Arm
- 基本要求：保持语义等价



编译技术和主要分支


源代码
单词流
语法解析树
解释执行
① 计算器
抽象语法树
类型推导/检查
线性IR
解释执行
② 虚拟机
JIT
汇编代码
可执行程序
③ 裸机执行

编译器基本框架


前端-理解程序
中间层-分析程序
后端-平台适配
源代码
词法分析
句法分析
生成虚拟语法树
类型推导／检查
生成线性IR
代码优化
指令选择
寄存器分配
后端优化
目标代码

词法分析：Lexical Analysis/ Tokenize

- 将字符串转换为单词流
- 现有理论/工具（如Flex）非常成熟，可直接使用

字符串： (1 + 2) * -3

单词流： <LPAR> <UNUM(1)> <ADD> <UNUM(2)> <RPAR> <MUL> <SUB> <UNUM(3)>

正则表达式声明词法

- 正整数：[1-9][0-9]*
- 无符号浮点数：[1-9][0-9]*(ϵ|.[0-9][0-9]*)
- 浮点数：（-|ϵ）[1-9][0-9]*(.[0-9][0-9]*|ϵ)
- 实际情况中，负号一般不在词法分析环节确定

句法分析：Parsing

- 分析单词流是否为该语言的一个句子（符合句法规则）

语法规则示例：

```
[1] E → E <ADD> E
[2]   | E <SUB> E
[3]   | E <MUL> E
[4]   | E <DIV> E
[5]   | E <EXP> E
[6]   | <LPAR> E <RPAR>
[7]   | NUM
[8] NUM → <UNUM>
[9]     | <SUB> <UNUM>
```

目标单词流：

<LPAR> <UNUM(1)> <ADD> <UNUM(2)>
<RPAR> <MUL> <SUB> <UNUM(3)>

解析过程：

规则	展开 E
[3]	⇨ E <MUL> E
[6]	⇨ <LPAR> E <RPAR> <MUL> E
[1]	⇨ <LPAR> E <ADD> E <RPAR> <MUL> E
[7]	⇨ <LPAR> NUM <ADD> E <RPAR> <MUL> E
[8]	⇨ <LPAR> UNUM <ADD> E <RPAR> <MUL> E
…	⇨ …

生成中间表示

- 进行上下文相关分析
  - 语法分析（词法+句法）不考虑上下文
  - 语法正确不一定整句有意义，如类型错误
- 生成抽象语法树（AST）
- 生成线性IR（LLVM IR）

示例：源代码->中间代码

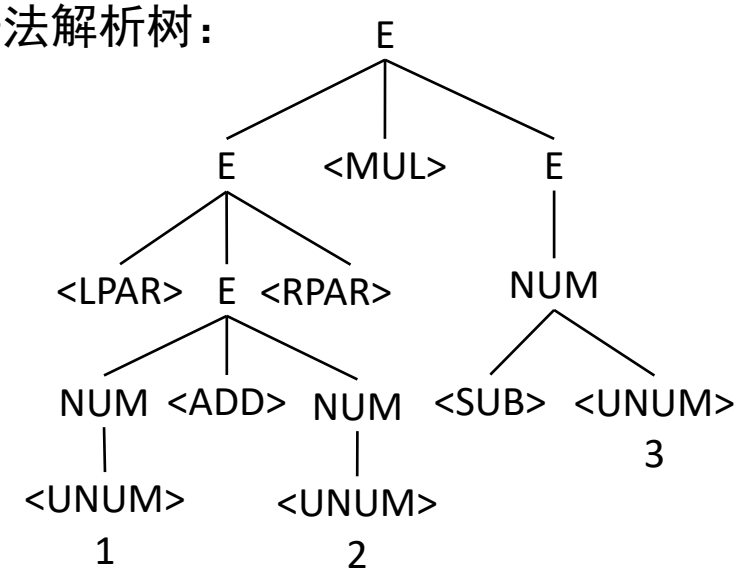
源代码：　　(1 + 2) * -3

1. 词法分析

<LPAR> <UNUM(1)> <ADD> <UNUM(2)> <RPAR> <MUL> <SUB> <UNUM(3)>

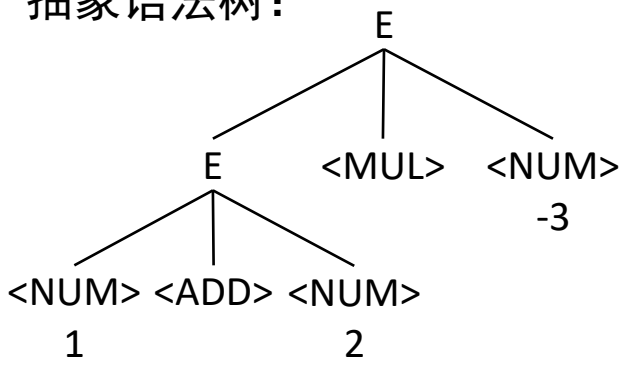
2. 句法分析

语法解析树：



3. 生成AST

抽象语法树：



3. 生成线性IR

线性IR：

```
t0 = 1 + 2;
t1 = -3;
t2 = t0 * t1;
```

代码优化

- 常量传导
- 循环优化
- 尾递归
- …

```
for (i=1; i<100; i++){
    int d = getInt();
    a = 2 * a * b * c * d;
}
```

优化

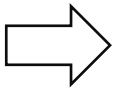
```
int t = 2 * b * c;
for (i=1; i<100; i++){
    int d = getInt();
    a = a * t * d
}
```

指令选择：Instruction Selection

- 将中间代码翻译为目标机器指令集

LLVM IR

```
BB1:
    %a = alloca i32
    %b = alloca i32
    %r = alloca i32
    store i32 1, i32* %a
    store i32 1, i32* %b
    %a1 = load i32, i32* %a
    %b1 = load i32, i32* %b
    %r1 = icmp eq i32 %a1, 0
    br i1 %r1, label %BB2, label %BB3
BB2:
    %a2 = add i32 %a1, %b1
    store i32 %a2, i32* %a
    br label %BB2
BB3:
    %a3 = load i32, i32* %a
    %r2 = add i32 %a3, %b1
    store i32 %r2, i32* %r
    ret void
```



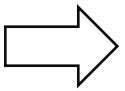
Arm汇编代码

```
main:
        sub     sp, sp, #16
        mov     %r1, #1
        str     %r1, [sp, #28]
        str     %r1, [sp, #24]
        ldr     %r2, [sp, #28]
        ldr     %r3, [sp, #24]
        cbnz    %r2, .LBB0_2
.LBB0_1:
        add     %r4, %r2, %r3
        str     %r4, [sp, #28]
        b       .LBB0_1
.LBB0_2:
        ldr     %r5, [sp, #28]
        add     %r6, %r3, %r5
        str     %r6, [sp, #20]
        add     sp, sp, #32
        ret
```

寄存器分配：Register allocation

- 指令选择假设寄存器有无限多，而实际寄存器数目有限
- 如果超出了寄存器数量需要将数据临时保存到内存中

```
main:
        sub     sp, sp, #16
        mov     %r1, #1
        str     %r1, [sp, #28]
        str     %r1, [sp, #24]
        ldr     %r2, [sp, #28]
        ldr     %r3, [sp, #24]
        cbnz    %r2, .LBB0_2
.LBB0_1:
        add     %r4, %r2, %r3
        str     %r4, [sp, #28]
        b       .LBB0_1
.LBB0_2:
        ldr     %r5, [sp, #28]
        add     %r6, %r3, %r5
        str     %r6, [sp, #20]
        add     sp, sp, #32
        ret
```



```
main:
        sub     sp, sp, #16
        mov     w8, #1
        str     w8, [sp, #28]
        str     w8, [sp, #24]
        ldr     w8, [sp, #28]
        ldr     w9, [sp, #24]
        cbnz    w8, .LBB0_2
.LBB0_1:
        add     w8, w8, w9
        str     w8, [sp, #28]
        b       .LBB0_1
.LBB0_2:
        ldr     w8, [sp, #28]
        add     w8, w9, w8
        str     w8, [sp, #20]
        add     sp, sp, #32
        ret
```

后端优化：

- 指令调度（Instruction Reordering）
- 窥孔优化（peephole optimization）
- …

假设：

指令	延迟
LDR	4
STR	1
ADD	1
SUB	1
MUL	3
SDIV	7
MOV	1
CBNZ	1

```
main:
        mov     w8, #1
        str     w8, [sp, #28]
        str     w8, [sp, #24]
        ldr     w8, [sp, #28]
        ldr     w9, [sp, #24]
        cbnz    w8, .LBB0_2
.LBB0_1:
        add     w8, w8, w9
        str     w8, [sp, #28]
        b       .LBB0_1
.LBB0_2:
        ldr     w8, [sp, #28]
        add     w8, w9, w8
        str     w8, [sp, #20]
        add     sp, sp, #32
        ret
```

优化 ⇨ ?

一些编译相关的概念和词汇

- 解释执行
- JIT（just-in-time compilation）
- AOT（ahead-of-time compilation）
- 静态类型
- 动态类型
- 运行时环境（RTE）
- no_std
- 垃圾回收器

练习

- 实现并验证运算符优先级解析算法
  - 考虑加入负数和括号？
  - 可以使用GPT