Lecture 1

课程入门

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：

崔漠寒
cuimohan@fudan.edu.cn
江湾校区交叉二号楼D6010

张业鸿
yehongzhang23@m.fudan.edu.cn
江湾校区交叉二号楼D4004

倪雯倩
22210240038@m.fudan.edu.cn
江湾校区交叉二号楼A4008

董方
fdong22@m.fudan.edu.cn
江湾校区交叉二号楼A2008

陈实力
20307110078@fudan.edu.cn

王兆瀚
20307130171@fudan.edu.cn

柏露
20307130208@fudan.edu.cn

课程信息

- 课堂教学：
  - 时间：星期五 2-4（5）节（8:55am-11:35am/12:30pm）[1-16周]
  - 地点：光华楼西508室（HGX508）
- 上机实践：
  - 时间：星期五9-10节（16:20am-18:00pm）[每双周]
  - 地点：H逸夫楼302
- 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
```

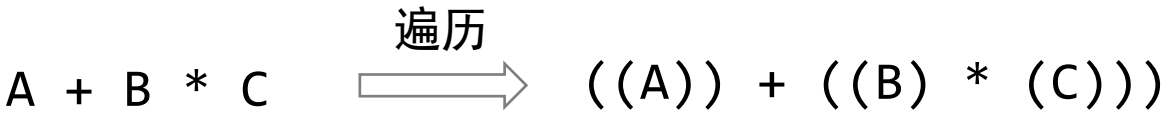
机器码

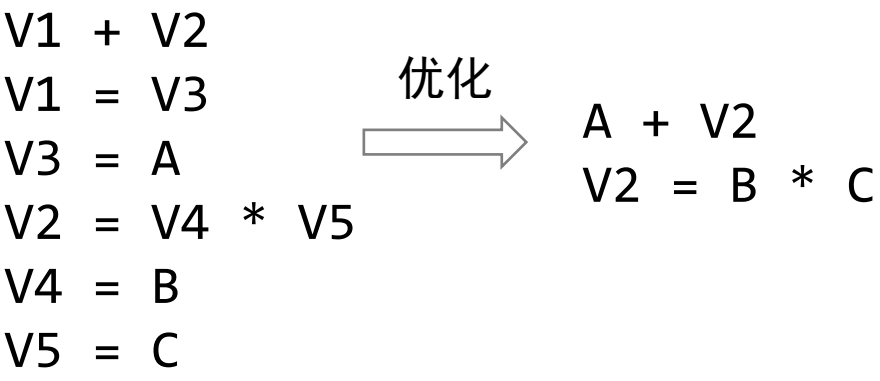
```
50 BF 04 20
40 00 E8 F5
FE FF FF 31
C0 59 C3 90
```

问：第一门被广泛使用的通用高级编程语言是？

Fortran（1954）如何实现自举？

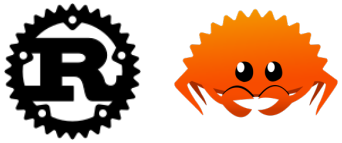
- 23500行汇编代码，耗费人力18人年
- 很多先进思想至今沿用，以操作符优先级遍历为例
  - 将+/-替换为))+((，或))-((
  - 将*//替换为)*(，或)/(
  - 在程序开头添加 ((，结尾添加))



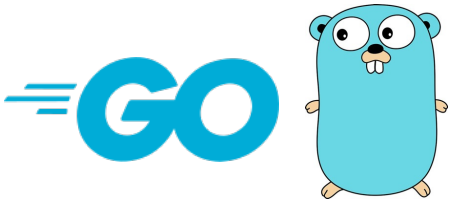


David Padua, "The fortran I compiler," *Computing in Science & Engineering* 2, no. 1 (2000): 70-75

新型编程语言层出不穷



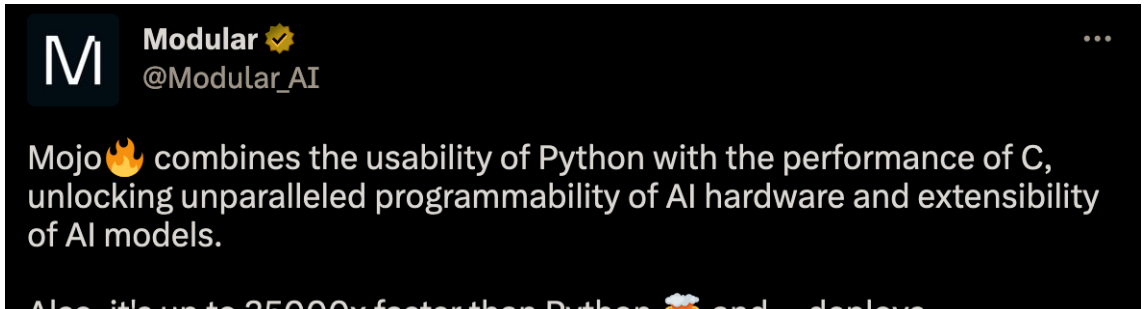
Mozilla（浏览器引擎）
Graydon Hoare
2006-2014（v1）



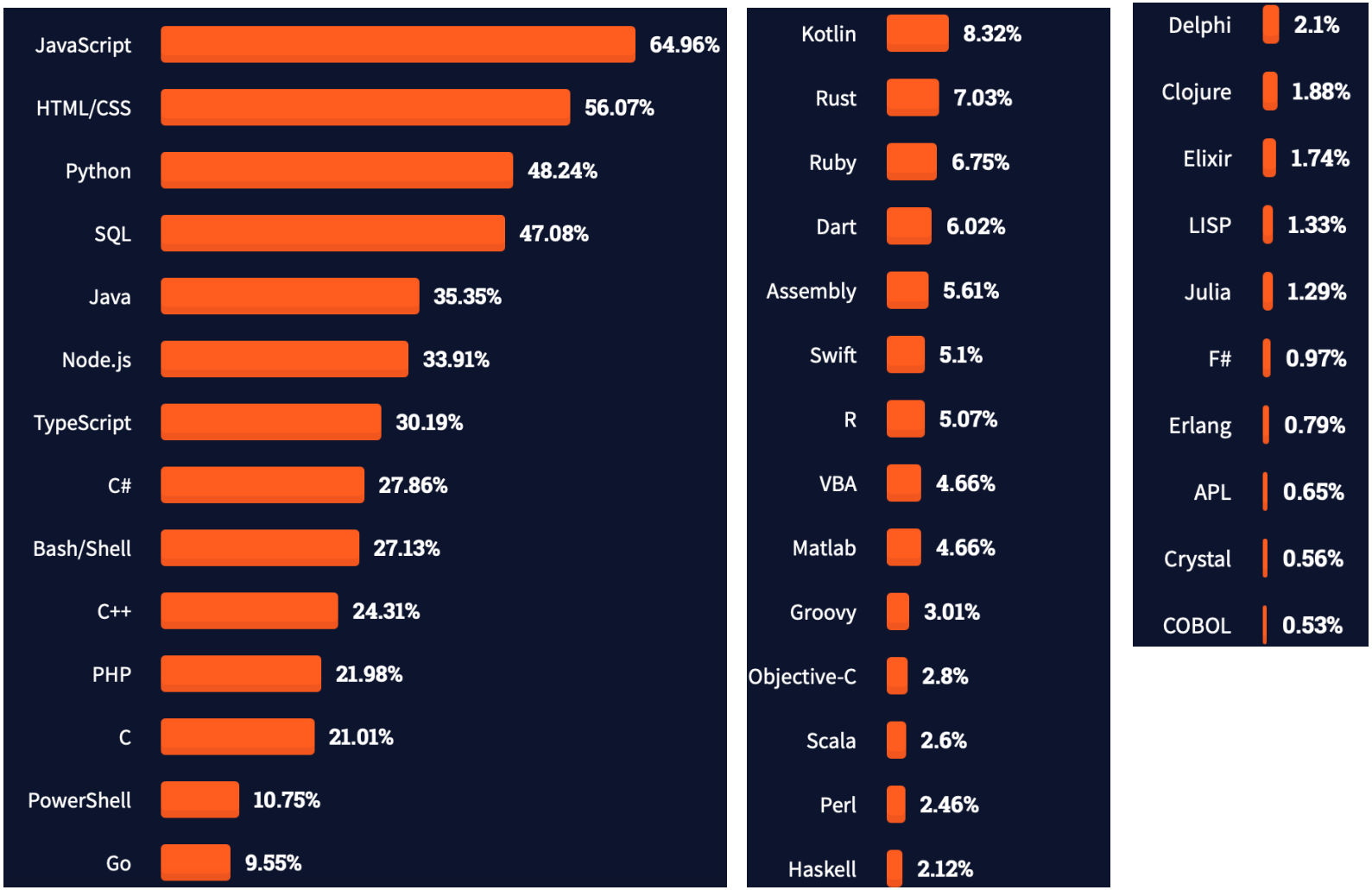
Google（多核、分布式服务）
R. Griesemer, R. Pike, K. Thompson
2007-2012（v1）

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

Modular（人工智能）
Chris Lattner
2022-2023（v?）



Stackoverflow语言使用统计排名



https://insights.stackoverflow.com/survey/2021#most-popular-technologies-language-prof

编译器和编程语言的重要性

- 计算机领域最高荣誉：ACM Turing Award

图灵奖得主

Liskov (subtype) substitution principle

2020年
Aho & Ullman

2008年
Barbara Liskov

面向对象
(Smalltalk语言)

BNF范式/ALGOL 60

Optimizating Compilers

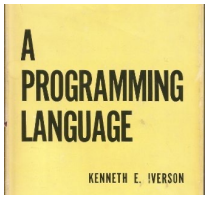
2003年
Alan Kay

2005年
Peter Naur

2006年
Frances Allen

$\{\emptyset\}P\{\psi\}$

Hoare Logic



FORTRAN
（语言）

ALGOL 60
（语言）

1980年
Tony Hoare

1979年
Kenneth Iverson

1977年
John Backus

1972年
Edsger Dijkstra

问：自然语言可以编程吗？

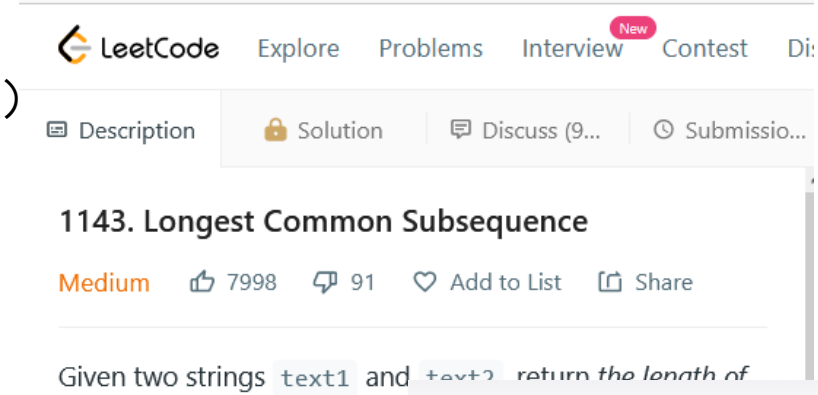
抽象级别

自然语言
（外祖母编程语言）

高级编程语言

汇编语言

机器码



LLM(GPT)好像可以？
正确性难以保证

To solve this problem, you can use dynamic programming. Here's the Python code to find the length of the longest common subsequence between two given strings:

python Copy code

```python
def longestCommonSubsequence(text1, text2):
    m, n = len(text1), len(text2)
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if text1[i - 1] == text2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1] + 1
            else:
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

    return dp[m][n]

# Test cases
text1_1, text2_1 = "abcde", "ace"
print(longestCommonSubsequence(text1_1, text2_1))  # Output: 3

text1_2, text2_2 = "abc", "abc"
print(longestCommonSubsequence(text1_2, text2_2))  # Output: 3
```

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

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。

434 人赞同了该回答

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。

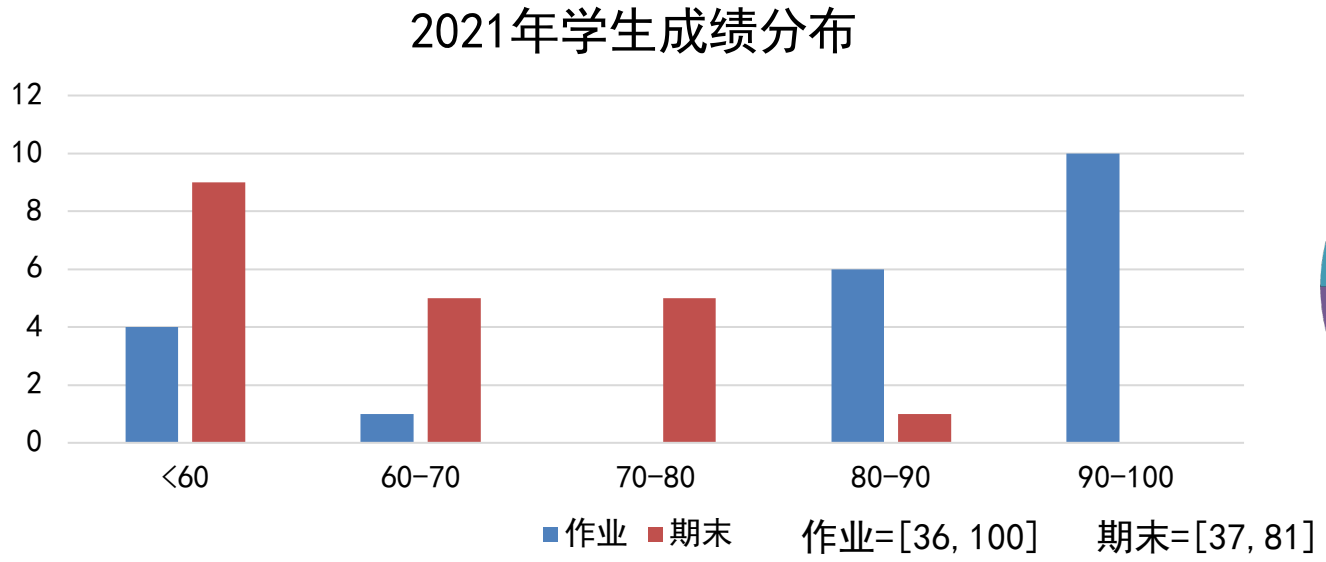
教学大纲设计（Tentative）

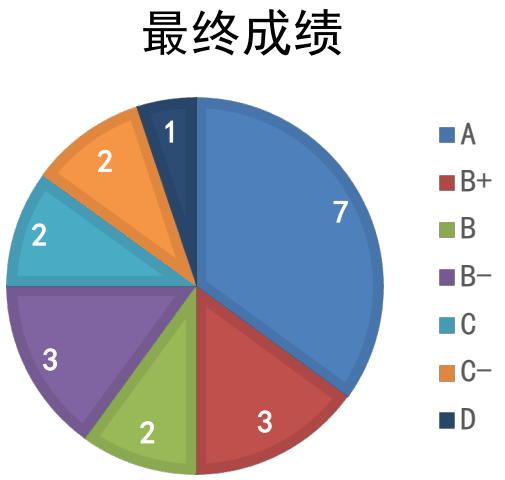
时间	周	授课内容	拔尖班	上机
Sep-8	1	课程入门	编程语言	
Sep-15	2	词法分析	词法设计	Calculator
Sep-22	3	CFG + LL(1)	句法设计-1	
Sep-29	4	LR(1) + More	句法设计-2	TeaPL Parser
Oct-6	5	国庆节		
Oct-13	6	AST + 类型检查（线上）	扩展语法	AST Gen + Type Check
Oct-20	7	线性IR + 解释执行	扩展语法	
Oct-27	8	SSA和优化（线上线下结合）	MemorySSA	Linear IR Gen
Nov-3	9	常用优化算法	指针分析	
Nov-10	10	了解高级类型系统	扩展语法	SSA + Optimization
Nov-17	11	指令选择	扩展指令	
Nov-24	12	指令调度	实践扩展	Instruction Selection
Dec-1	13	寄存器分配	实践扩展	
Dec-8	14	后端集成与优化	实践扩展	Register Allocation
Dec-15	15	并行优化（线上）	前沿扩展	
Dec-22	16	调试和异常处理 + 复习	扩展语法	Integration

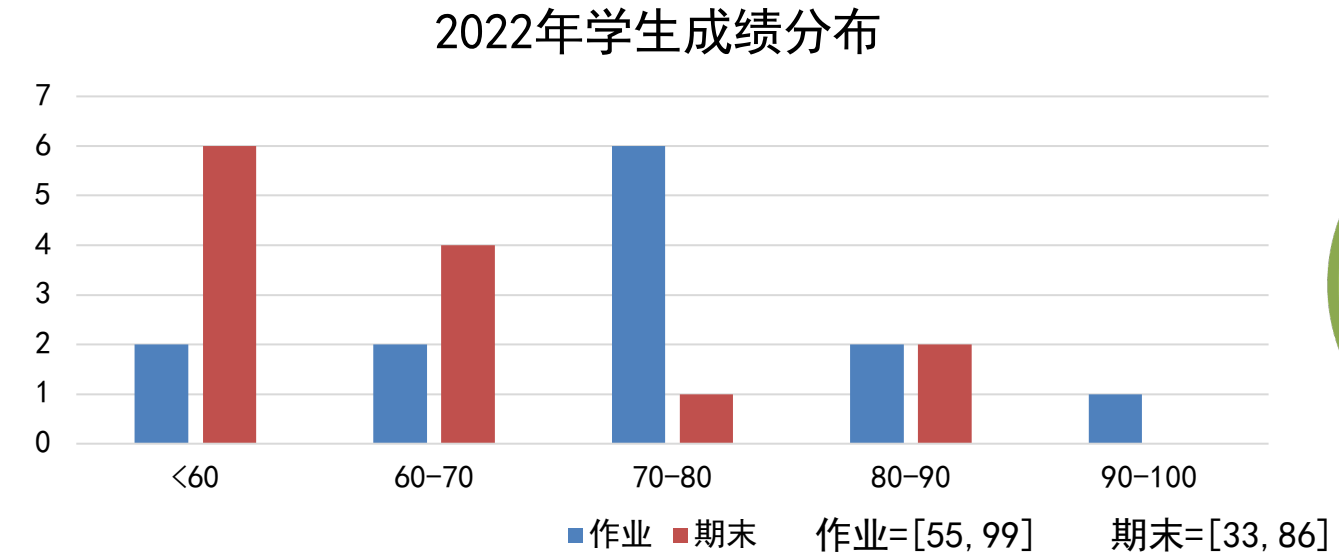
课程考核

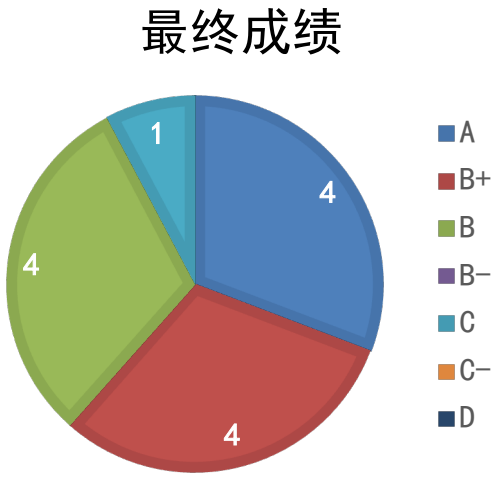
- 课程作业：50%
  - 8次上机实验：
    - 分数占比：4% + 6% + 7% + 7% + 7% + 7% + 7% + 5%;
  - 分组：
    - 普通班：原则上2人/组
    - 拔尖班：独立完成
- 开卷考试：50%
  - 拔尖班：多1道附加题
  - 2024-01-05 15:30~17:30

往年学生成绩


2021年学生成绩分布
12
10
8
6
4
2
0
<60
60-70
70-80
80-90
90-100
作业
期末
作业=[36, 100]
期末=[37, 81]

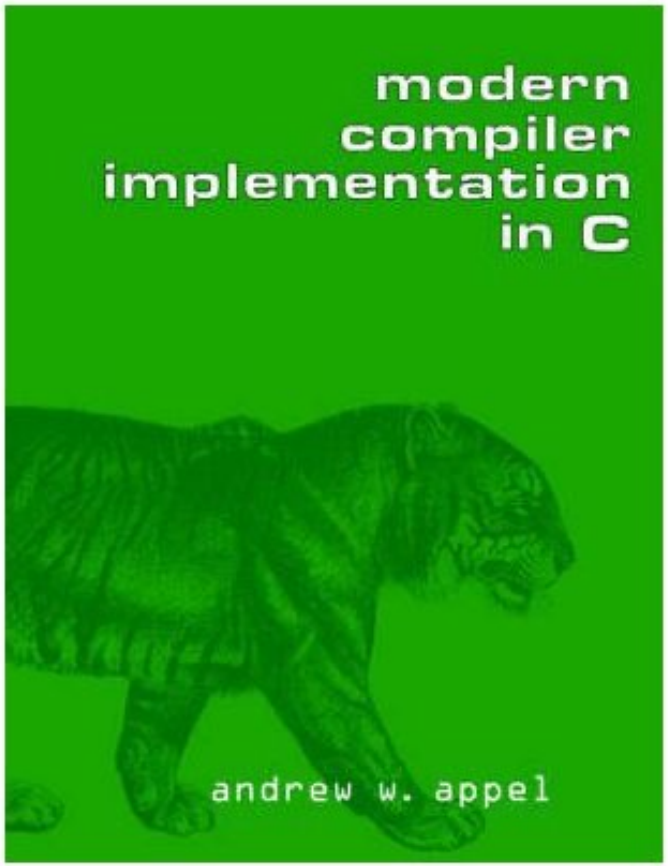

最终成绩
7
3
2
3
2
2
1
A
B+
B
B-
C
C-
D

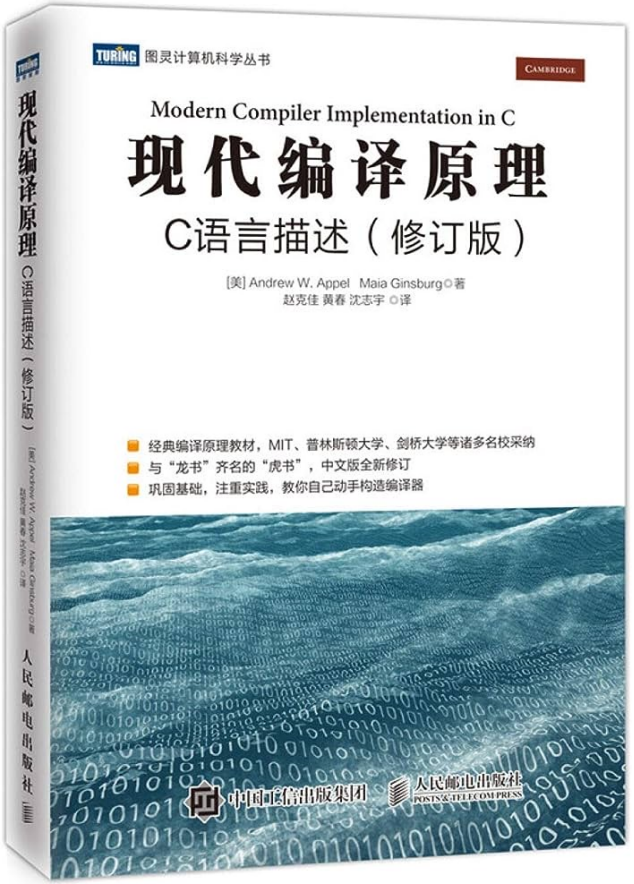

2022年学生成绩分布
7
6
5
4
3
2
1
0
<60
60-70
70-80
80-90
90-100
作业
期末
作业=[55, 99]
期末=[33, 86]


最终成绩
4
4
4
1
A
B+
B
B-
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

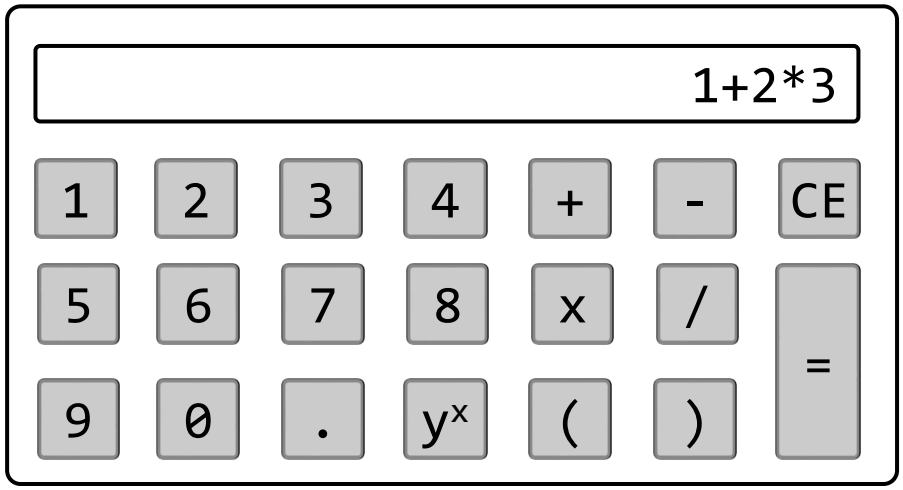




二、编译：以计算器为例

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: input;
Output: output; //保存解析结果

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

难点：如何区分“-”

```
Input: in;
Output: tok; //保存解析结果

while (true) {
    cur = in.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 ✓

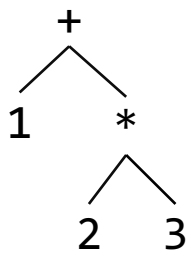
负号问题可以解决了

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

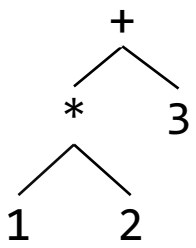
1*2+3

(1*2)+3



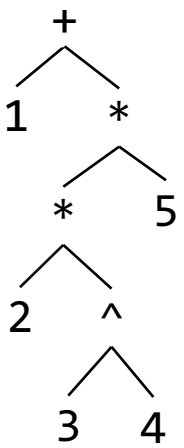
1+2*3

1+(2*3)



1+2*3^4*5

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

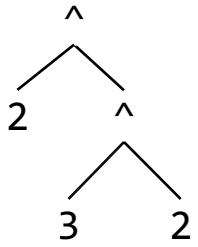


Fortran方法：(((1)))+(((2))*((3)^(4))*((5)))

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



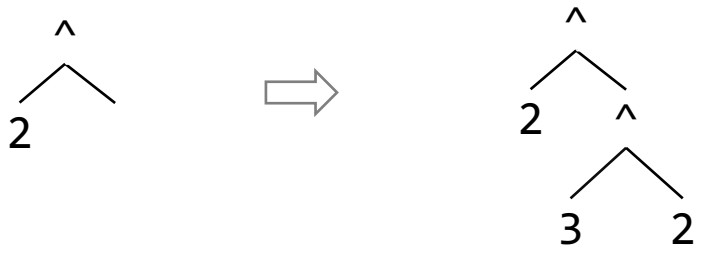
2^(3^2)

(((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

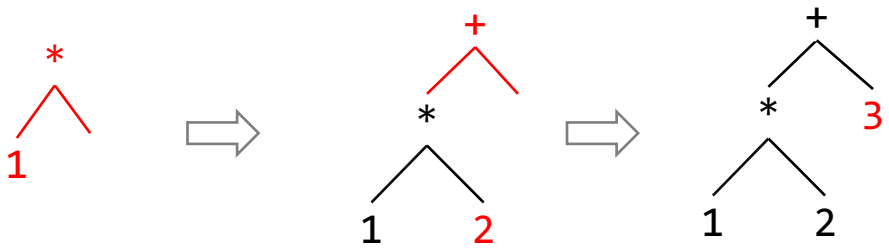


1+3^2

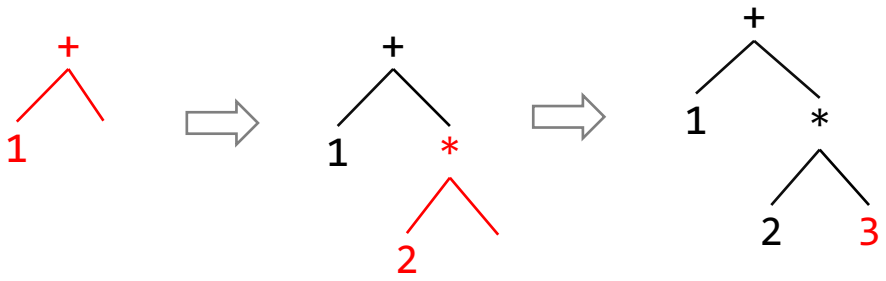
运算符优先级解析思路

- 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

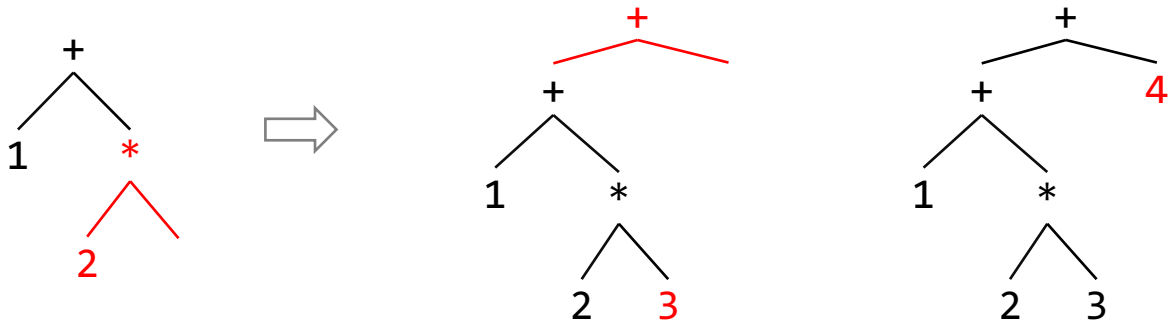


1+2*3

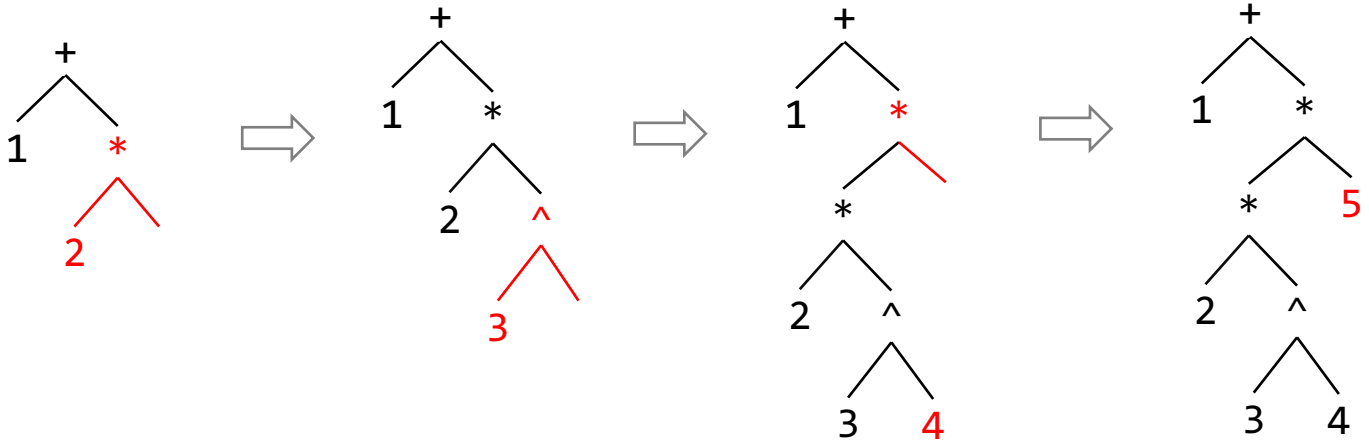


运算符优先级解析思路

1+2*3+4



1+2*3^4*5

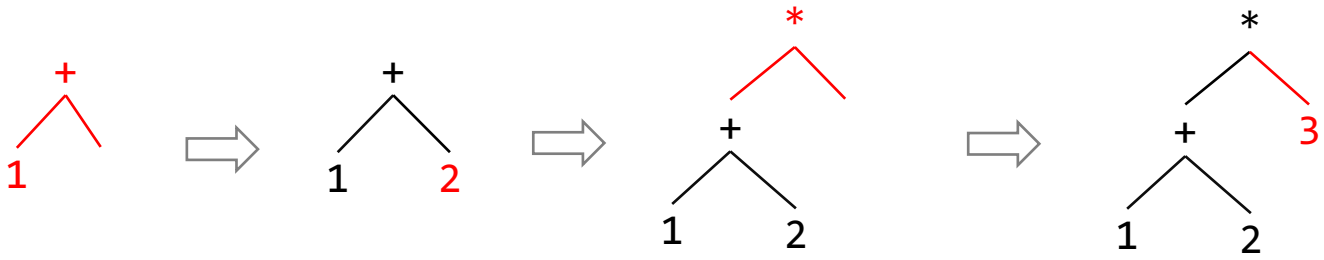


如果当前运算符≤栈顶运算符的优先级：

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

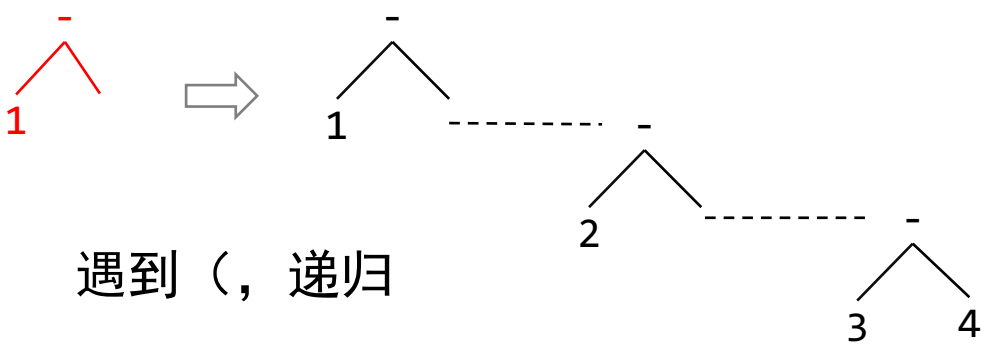
运算符优先级解析思路

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



遇到（，进栈，遇到），将操作数加到树上?

(1-(2-(3-4)))



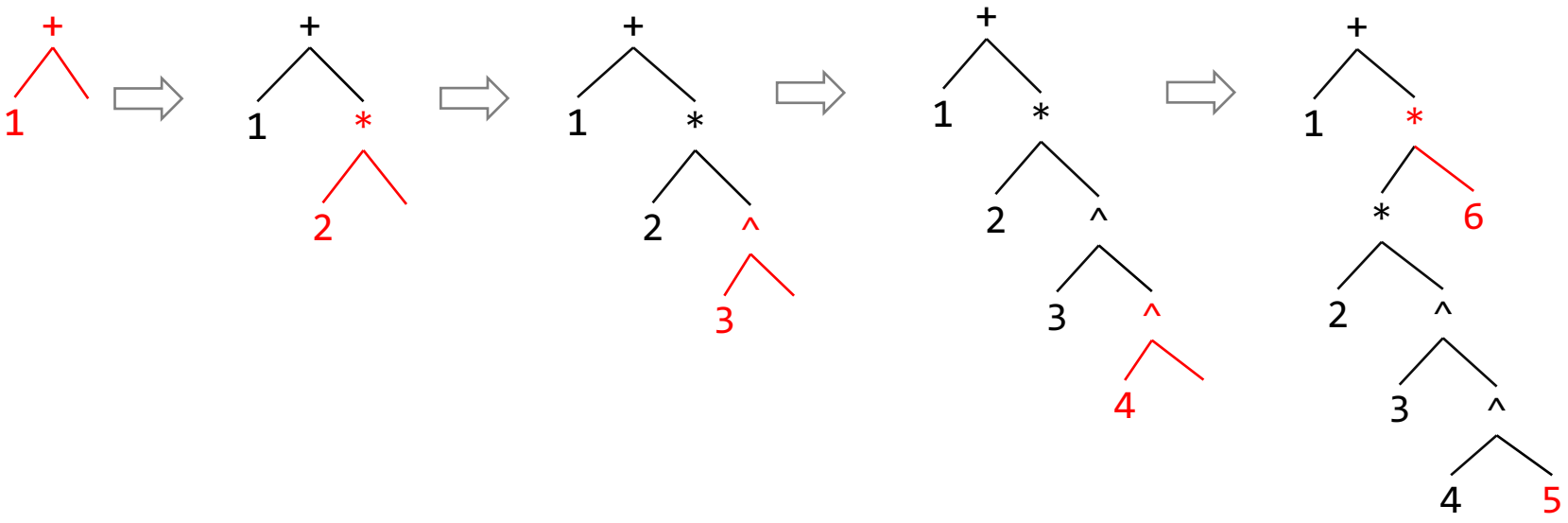
遇到（，递归

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
```

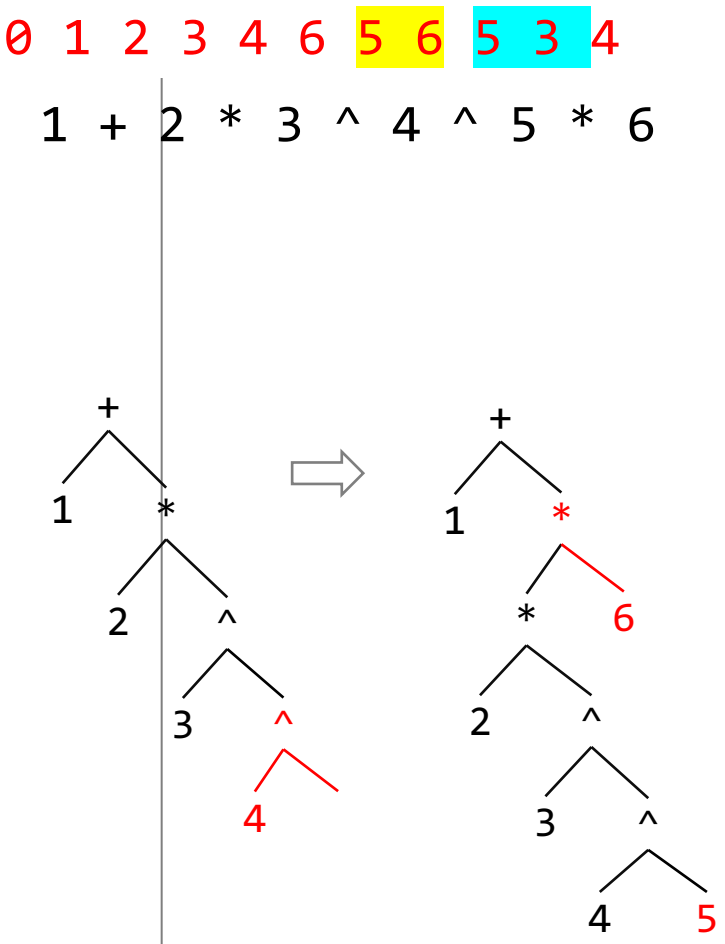
初始化优先级 0 1 2 3 4 6 5 6 5 3 4

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

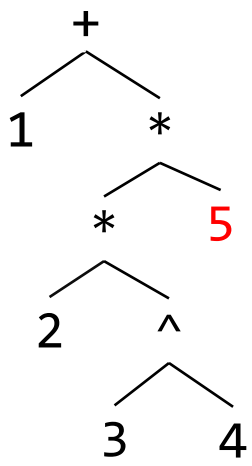


算法实现参考：加入括号

```
Parse(cur, precedence) -> BinTree {
    root = null;
    cur = token.next();
    if tok.type == tok::LPAR
        cur = token.next();
        root = Parse(cur, 0);
    while true:
        peek = cur.peek();
        if peek.tokentype == tok::RPAR
            break;
        if peek.tokentype != tok::BINOP
            return -1;
        lp, rp = Preced[peek];
        if lp < precedence
            break;
        cur.next();
        right = Parse(cur, rp)
        root = CreateBinTree(peek, cur, right)
    return root;
}
```

步骤3：解释执行/翻译为逆波兰表达式

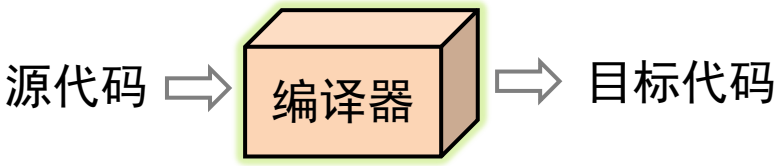
- 先序遍历语法解析树=>波兰表达式
  - + 1 * * 2 ^ 3 4 5
  - 满二叉树无歧义
- 后序遍历语法解析树=>逆波兰表达式
  - 1 2 3 4 ^ * 5 * +
- 逆波兰表达式方便计算：
  - 顺序读取，遇到操作数则入栈
  - 遇到运算符，则弹出栈顶的两个操作数，求值后将结果入栈



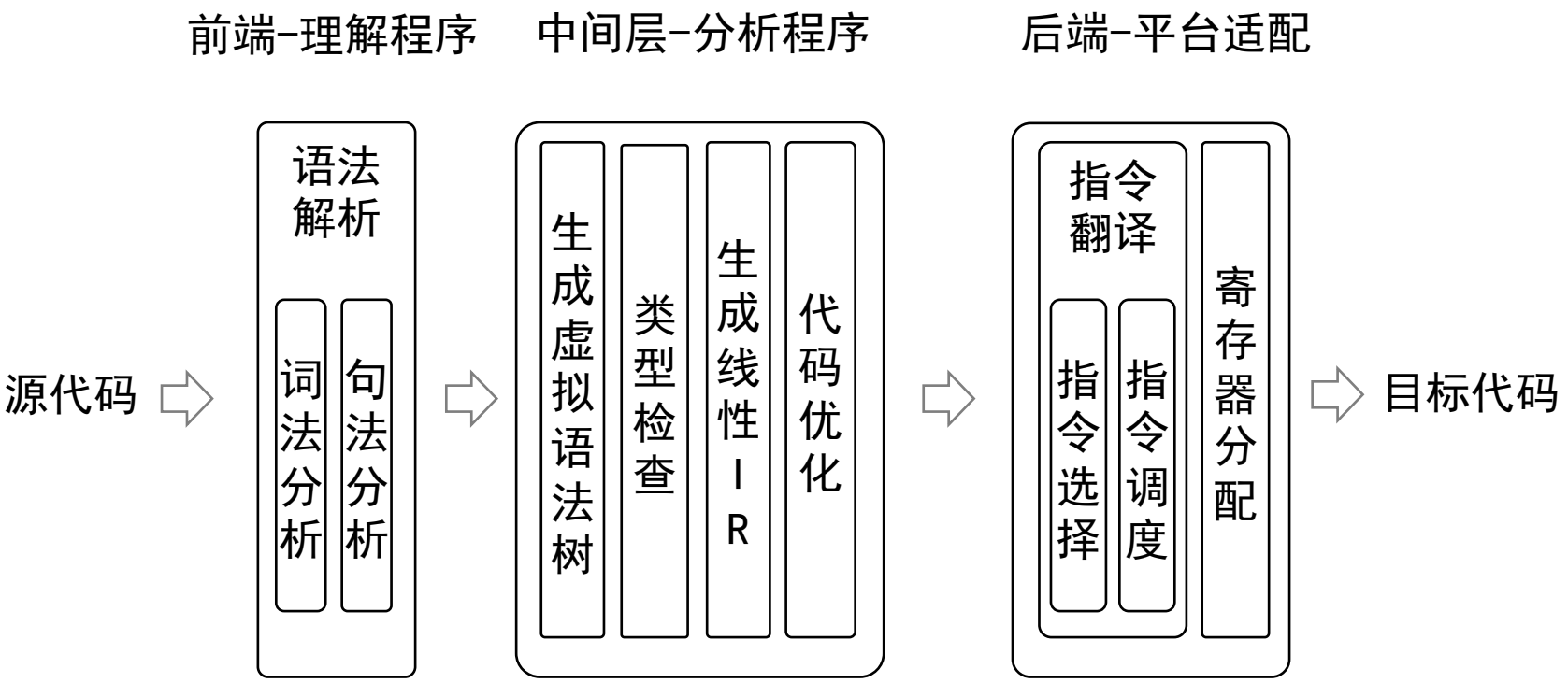
三、编译流程概览

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
语法解析
词法分析
句法分析
生成虚拟语法树
类型检查
生成线性IR
代码优化
指令翻译
指令选择
指令调度
寄存器分配
目标代码

词法分析：Lexical Analysis/ Tokenize

- 将字符串（char stream）转换为单词流（token stream）
- 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

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>
```

目标句法：　<LPAR> <NUM(1)> <ADD> <NUM(2)> <RPAR> <MUL> <SUB> <NUM(3)>

句法解析：

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

… ⇨ …

生成中间代码（语法制导）

- 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

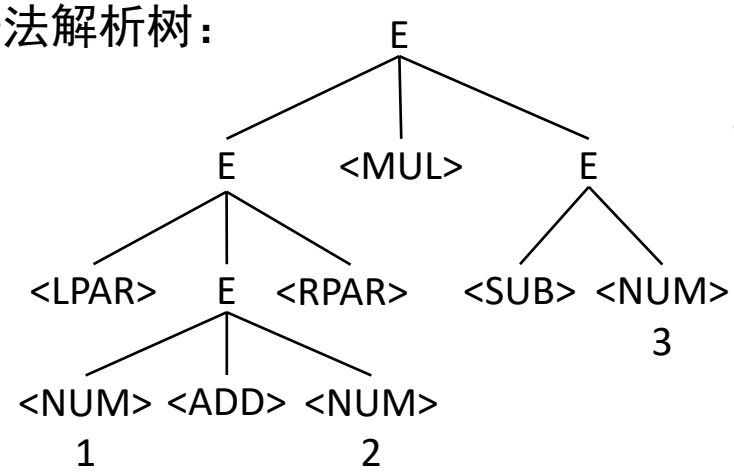
源代码：　　(1 + 2) * -3

1. 词法分析

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

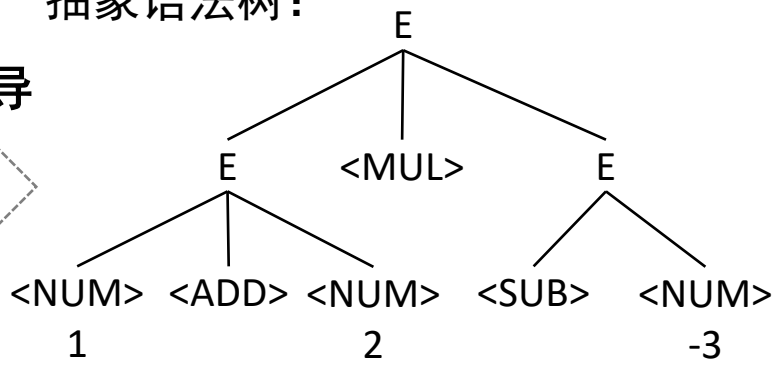
2.句法分析

语法解析树：



3.语法制导

抽象语法树：



3. 语法制导

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
  - 考虑函数调用规约等情况

IR

```
define dso_local i32 @ident(i32 %0) #0 {
  %2 = load i32, i32* @global_var, align 4
  %3 = add nsw i32 %0, %2
  %4 = mul nsw i32 %3, 2
  ret i32 %4
}
```

汇编代码

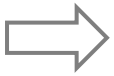
```
PUSH    %rbp
MOV     %rsp, %rbp
MOV     %edi, -4(%rbp)
MOV     -4(%rbp), %eax
ADD     global_var, %eax
SHL     $1, %eax
MOV     %eax, -8(%rbp)
MOV     -8(%rbp), %eax
POP     %rbp
RET
```

指令调度：Instruction Reordering

- 根据计算性能瓶颈优化指令顺序，假设：
  - 特定指令消耗固定的时钟周期：1：ADD/2：MOV/3：MUL/7：DIV
  - 后一条指令的操作数可用时会进入下一条指令，无需等待

开始	结束	指令
1	2	MOV $-12(%rsp), r1
2	3	MOV $-16(%rsp), r2
4	4	ADD r2, r1
5	6	MOV $-20(%rsp), r2
6	7	MOV $-24(%rsp), %eax
8	14	DIV r2,
15	16	MOV $-28(%rsp), r2
17	19	MUL r1, r2
18	19	MOV %eax, $-24(%rsp)
20	21	MOV r2, $-28(%rsp)

优化



开始	结束	指令
1	2	MOV $-20(%rsp), r1
2	3	MOV $-24(%rsp), %eax
4	10	DIV r1
5	6	MOV $-12(%rsp), r1
6	7	MOV $-16(%rsp), r2
8	8	ADD r2, r1
9	10	MOV $-28(%rsp), r2
11	13	MUL r1, r2
12	13	MOV %eax, $-24(%rsp)
13	14	MOV r2, $-28(%rsp)

寄存器分配：Register allocation

- 如何使用数量最少的寄存器？
  - 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
  - 通过寄存器分配降低数据存取开销

```
MOV 0, r1
MOV 1, r2
ADD r1, r2
ADD r2, r3
ADD r3, r4
```

```
MOV $0, %eax
MOV $1, %edx
ADD %edx, %eax
ADD %eax, %edx
ADD %edx, %eax
```

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