CS30017 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
- 助教：

江湾校区 交叉二号楼D6023
xuh@fudan.edu.cn

孙一
suny21@m.fudan.edu.cn

崔晨昊
chcui23@m.fudan.edu.cn

课程信息

- 课堂教学：
  - 时间：星期五 6-8节（1:30pm-4:10pm）[1-16周]
  - 地点：第3教学楼306教室（H3306）
- 上机实践：
  - 时间：星期五9-10节（4:20am-6:00pm）[1-16周]
  - 地点：H逸夫楼202、204、205
- 课程平台：
  - 课程主页：https://tea-compiler.github.io
  - 官方平台：Elearning
  - 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
- AI时代是否还需要学习编译原理？改革方向？
  - 我也在思考这个问题

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

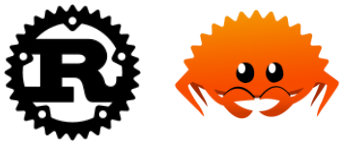
```
A + B * C   ⇒   ((A)) + ((B) * (C)))
```

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, et al
2007-2012（v1）

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

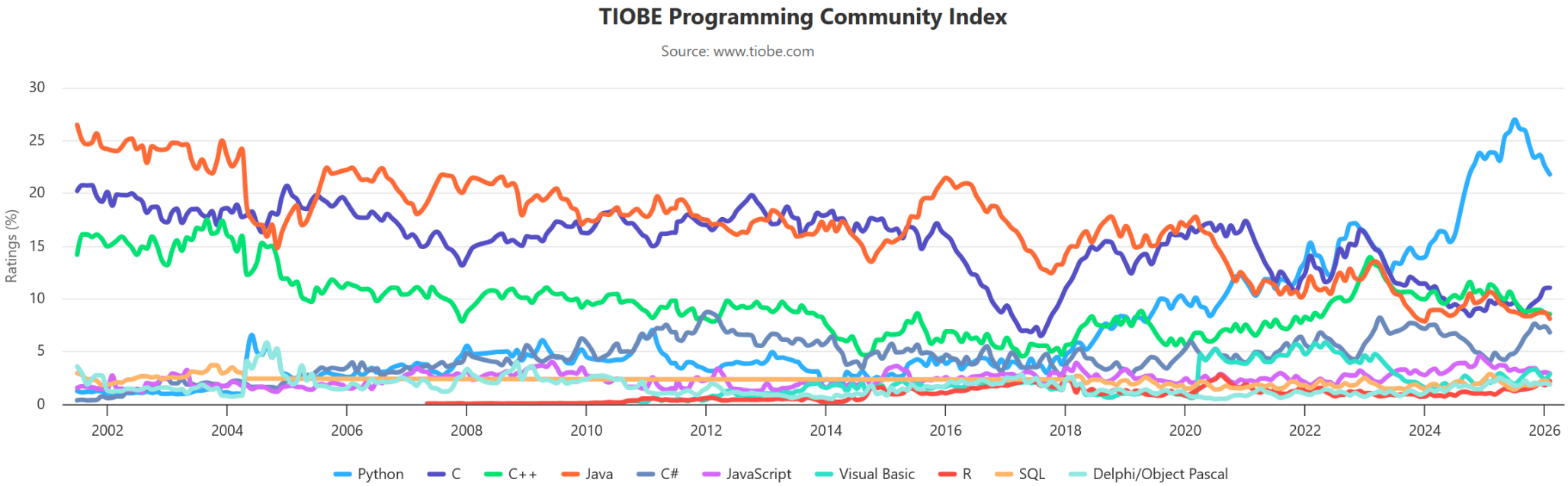
基于Python的GPU Kernel编程：

Triton

OpenAI

nVidia

TIOBE语言使用统计排名



TIOBE语言使用统计排名

Feb 2026	Feb 2025	Programming Language	Ratings	Change
1	1	Python	21.81%	-2.08%
2	4	C	11.05%	+1.22%
3	2	C++	8.55%	-2.82%
4	3	Java	8.12%	-2.54%
5	5	C#	6.83%	2.715%
6	6	JavaScript	2.92%	-0.85%
7	10	Visual Basic	2.85%	+0.81%
8	15	R	2.19%	1.14%
9	7	SQL	1.93%	-0.93%
10	9	Delphi/Object Pascal	1.88%	-0.29%
11	30	Perl	1.67%	+1.19%
12	11	Fortran	1.64%	-0.12%
13	14	PHP	1.34%	+0.20%
14	13	Rust	1.32%	-0.14%
15	12	Scratch	1.30%	-0.25%
16	8	Go	1.23%	-1.03%

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

type inference

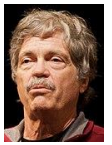
1991年
Robin Milner

Pascal

1984年
Niklaus Wirth

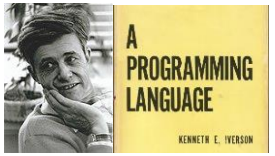
OOP: Simula

2001
Dahl & Nygaard

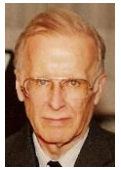


OOP: Smalltalk

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

自然语言编程 => 高级语言/汇编

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

LeetCode Explore Problems Interview New Contest Di

Description Solution Discuss (9... Submissio...

1143. Longest Common Subsequence

Medium 7998 91 Add to List Share

Given two strings `text1` and `text2`, return *the length of their longest* ***common subsequence***. If **common subsequence**, return `0`.

A **subsequence** of a string is a new stri the original string with some character deleted without changing the relative remaining characters.

- For example, `"ace"` is a subseq

A **common subsequence** of two string that is common to both strings.

Example 1:

```
Input: text1 = "abcde", text2 =
Output: 3
Explanation: The longest common
"ace" and its length is 3.
```

LLM好像可以？
如何保证正确性？

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

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。

知乎用户

编程话题下的优秀答主

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

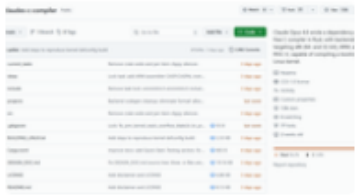
這是個好問題, 我光是發現怎麼學習編譯原理就花了不少時間, 也買了不少書, 但每本書的實作都不同, 讓學習更難了。

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。

知乎上的讨论

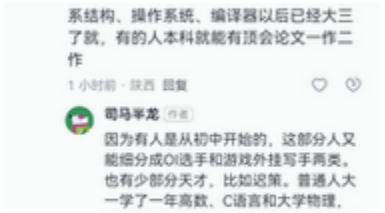
如何看待Anthropic用纯AI实现的CCC编译器可编译Linux?



酱紫君：github 上玩具编译器更是浩如烟海。 真正的圣杯是编译优化，世界上只有一个 LLVM，连 Rust 都不敢扔了 阅读全文

▲ 赞同 1458 ▼ 100 条评论 02-24

计算机本科生花大量时间写编译器，操作系统是不是不务正业?



知乎用户：很多独角兽在开发比如针对深度学习的编译器，在设计算子优化等方面，你会发现你还是要学习 AI 并把它结合到编译器里，而你可能甚至没看过 LLVM 阅读全文

▲ 赞同 1831 ▼ 106 条评论 2025-04-21

AI 都能写代码了，还要学计算机吗?

陆坚：”但没有人会质疑我们为什么要学读写、数学、科学和历史——尽管这些领域 AI 也一样做得很好。AI 确实会改变编程的方式 阅读全文

▲ 赞同 133 ▼ 22 条评论 2025-0

如何看待现阶段坚持不用 AI 的 「古法编程」?

飞天红猪侠：我发现了一些古法编程中不易察觉的优点，在此补充： 1. 古法编程使我的大脑更专注 古法编程更容易进入做产品的状态 阅读全文

▲ 赞同 5126 ▼ 346 条评论 2025-09-19

课程安排（Tentative）

周	时间	授课内容	上机内容	
1	Mar-6	课程入门	Rust	
2	Mar-13	词法分析	Rust	
3	Mar-20	上下文无关文法	Pest	布置HW1
4	Mar-27	自顶向下分析	答疑	
5	Apr-3	自底向上分析	答疑	
6	Apr-10	课程小结与研讨		验收HW1
7	Apr-17	类型推导		布置HW2
8	Apr-24	线性中间代码		布置HW3
9	May-1	假期		
10	May-8	静态单赋值形式	答疑	
11	May-15	过程内优化	答疑	
12	May-22	特邀讲座		验收HW2
13	May-29	课程小结与研讨		验收HW3
14	Jun-5	指令选择		布置HW4
15	Jun-12	寄存器分配	答疑	
16	Jun-19	后端优化		验收HW4

课程考核

- 平时成绩：50%
  - 4次实验：各占比10%
  - 课程研讨：10%
- 开卷考试：50%

实验选题

- 基于Tea语言（Rust语法的子集）
- 提供支持该子集的编译器框架（纯Rust实现）
  - https://github.com/tea-compiler/teac
- 学生扩展功能：
  - f32类型
  - 3选1：多维数组、for控制流、impl结构体方法

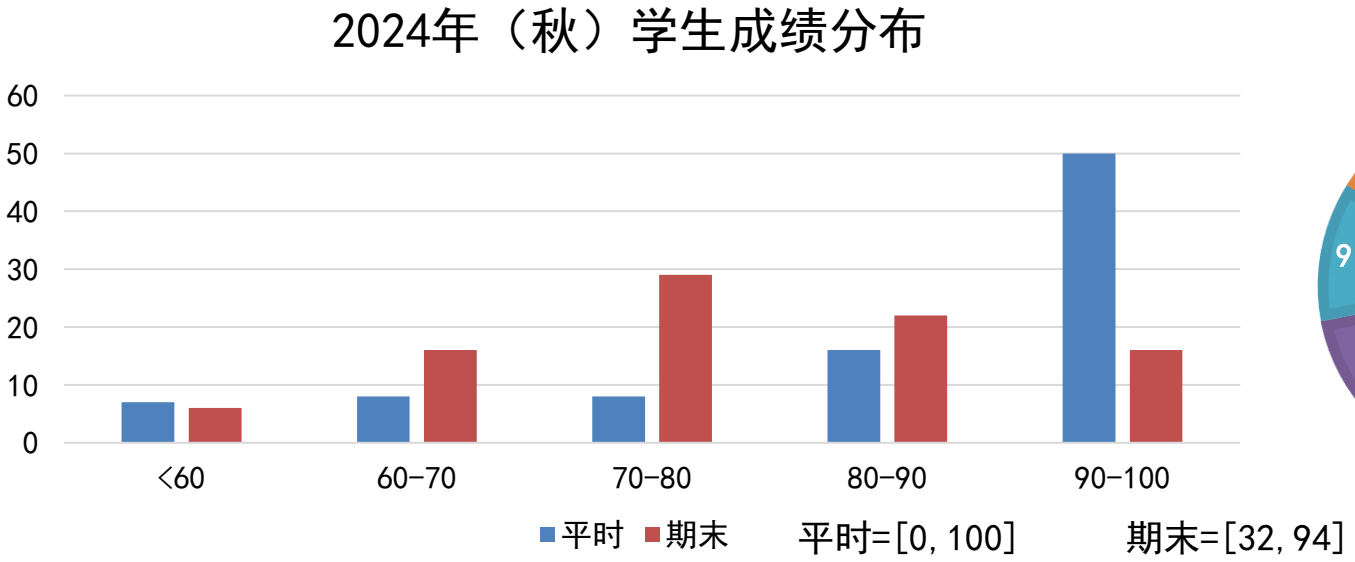
为什么选择Rust

- 新时代的系统编程语言：
  - 先进的特性：语法、CI/CD、包管理等
  - 更安全：减少了内存安全问题
  - 更可靠：可编译即可执行，基本不会遇到难调试的segmentation fault问题
- Rust很难上手？
  - AI时代大大降低了学习难度
  - 阅读C/C++代码的能力 => 阅读Rust代码的能力

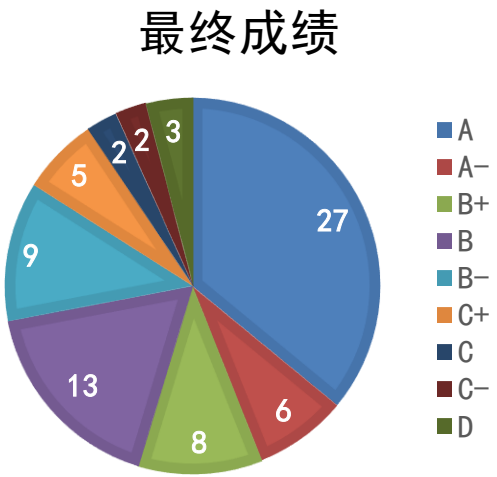
严禁作弊行为！！！

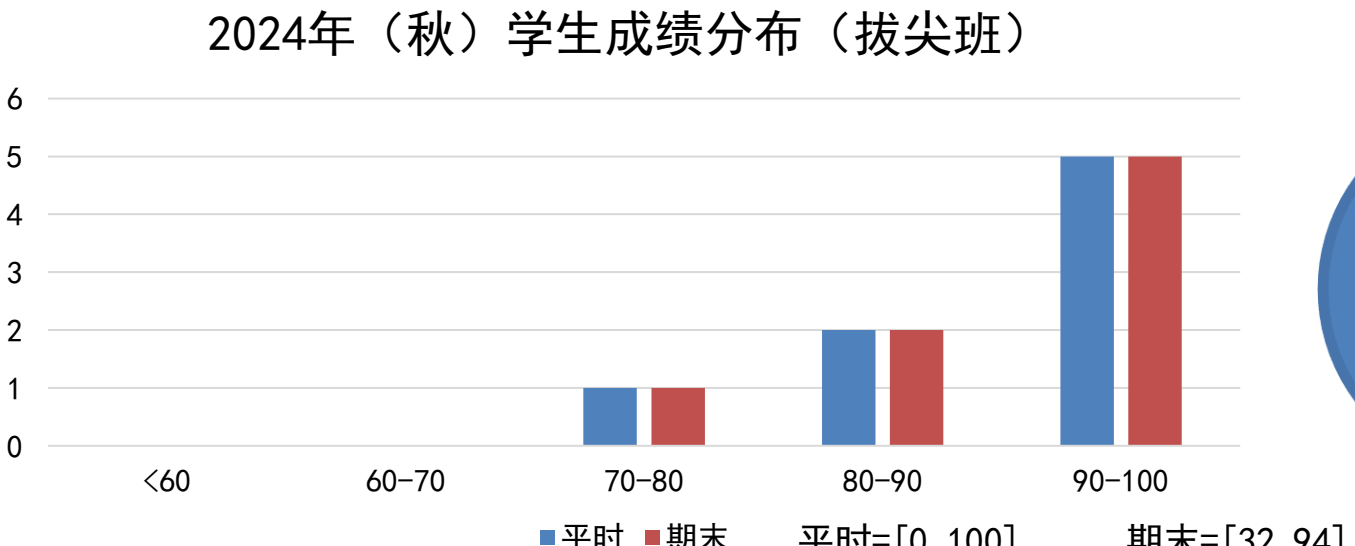
- 直接抄袭同学或GitHub网站上的答案。
- 使用AI编写代码，但无法解释代码的含义。

往年学生成绩

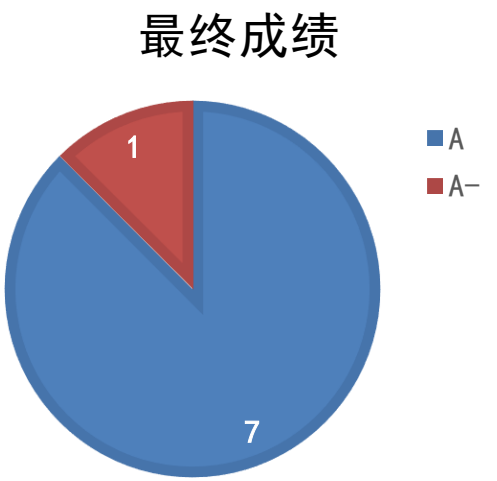


平时=[0, 100]　　期末=[32, 94]

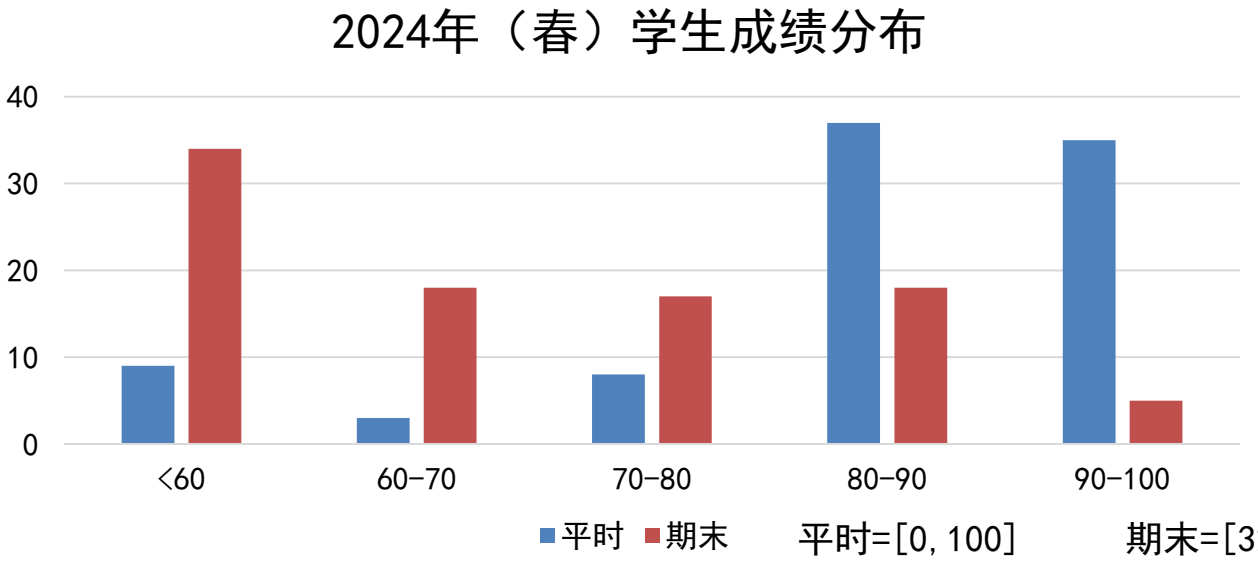




平时=[0, 100]　　期末=[32, 94]

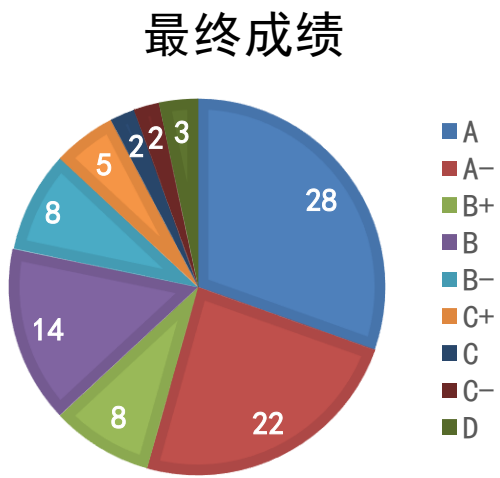


往年学生成绩



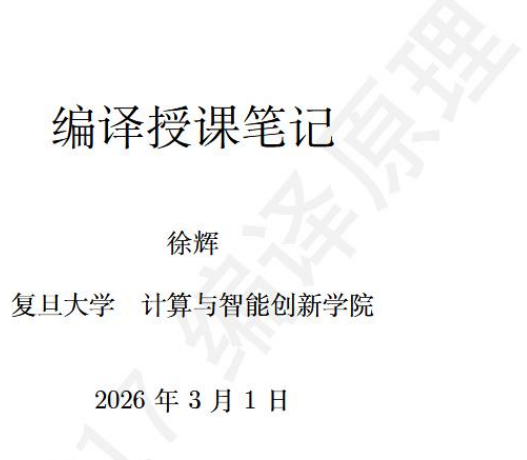
平时=[0, 100]

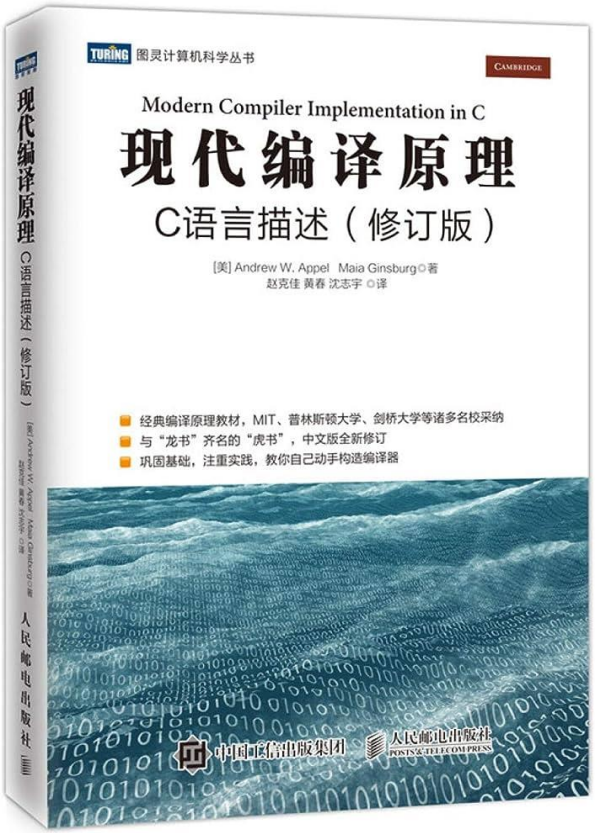
期末=[32, 94]



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

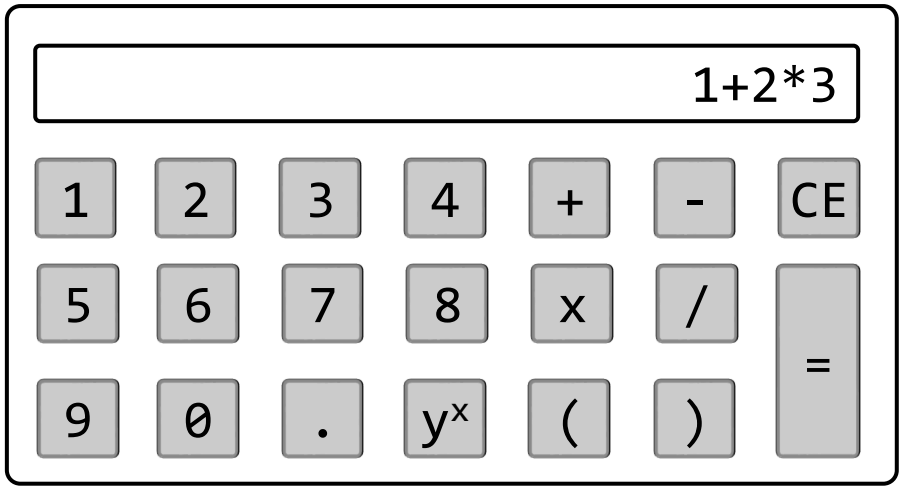




二、编译：以计算器为例

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 ✓

1**2+3 ✗

1+2*3^4*5 ✓

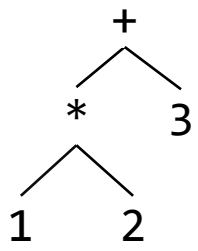
1+((2+3) ✗

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

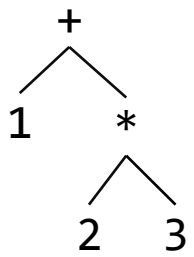
`1*2+3`

`(1*2)+3`



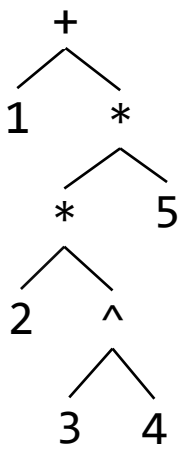
`1+2*3`

`1+(2*3)`



`1+2*3^4*5`

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



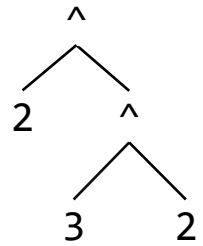
Fortran方法：`(((1)))+(((2))*((3)^(4))*((5)))` ✓

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)

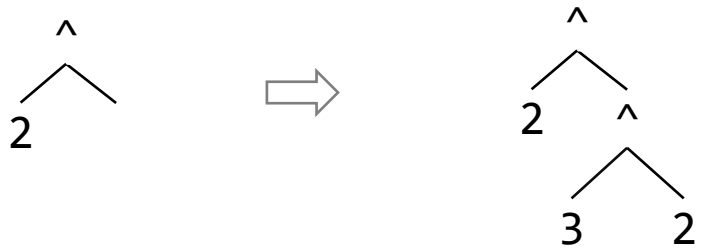


(((2)^(3)^(2)))

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

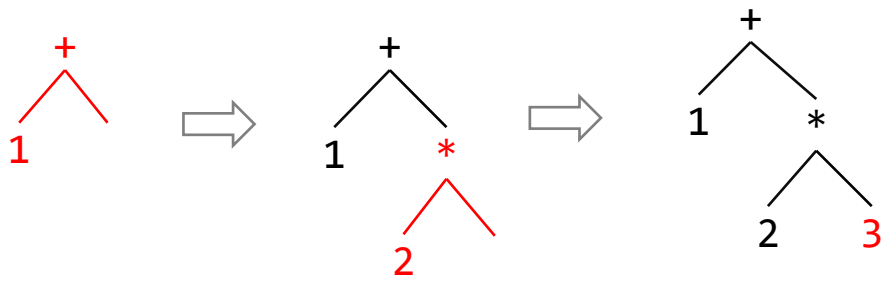


1+3^2

运算符优先级解析思路

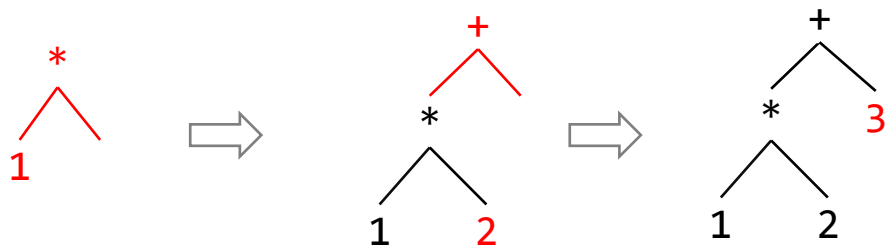
- 如果当前遇到的运算符为左结合
  - 如果当前运算符>栈顶运算符的优先级，将其作为右子节点

1+2*3



  - 如果当前运算符≤栈顶运算符的优先级，将其作为父节点?

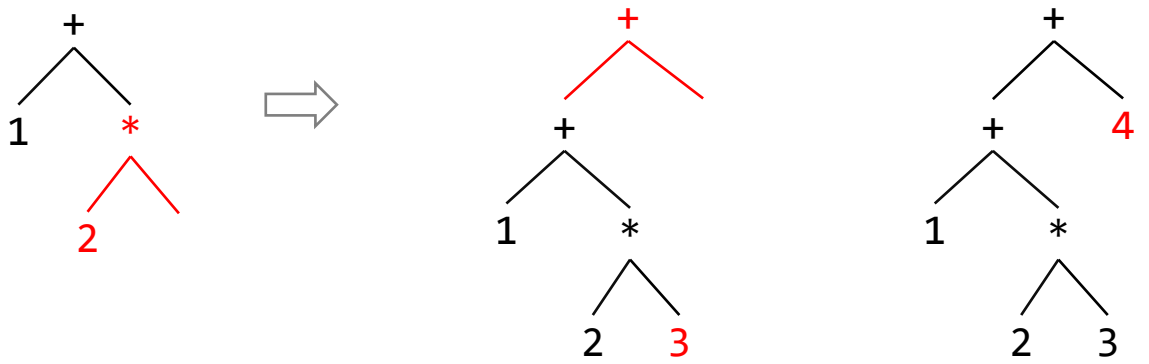
1*2+3



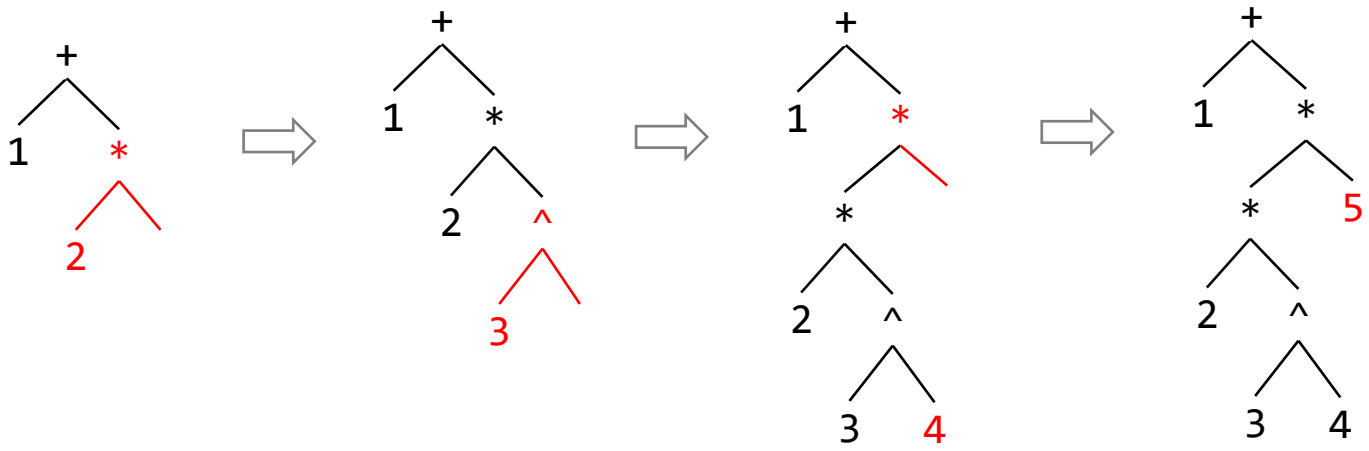
运算符优先级解析思路

- 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~~
  - 退栈直到当前运算符 >栈顶运算符，将其作为栈顶运算符的右子节点

1+2*3+4

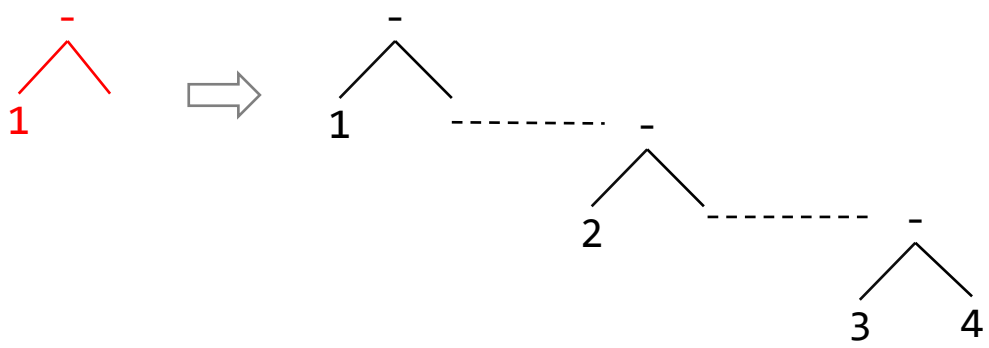


1+2*3^4*5

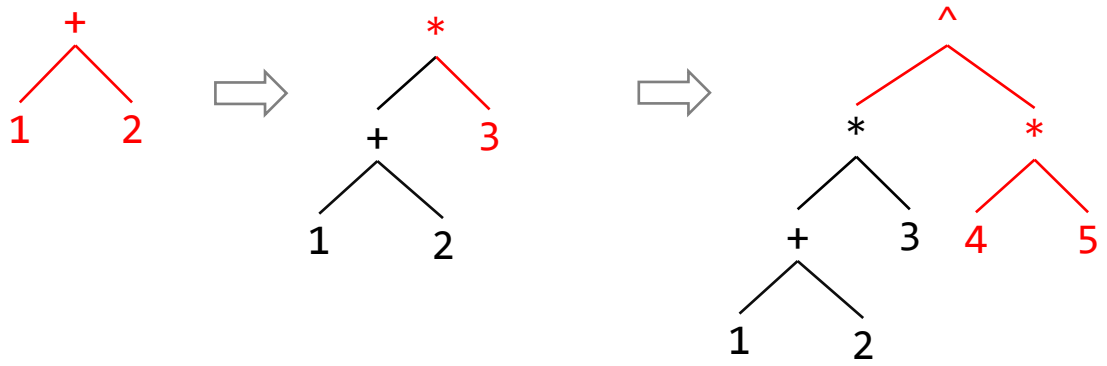


运算符优先级解析思路

(1-(2-(3-4)))　　遇到“（”，递归



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

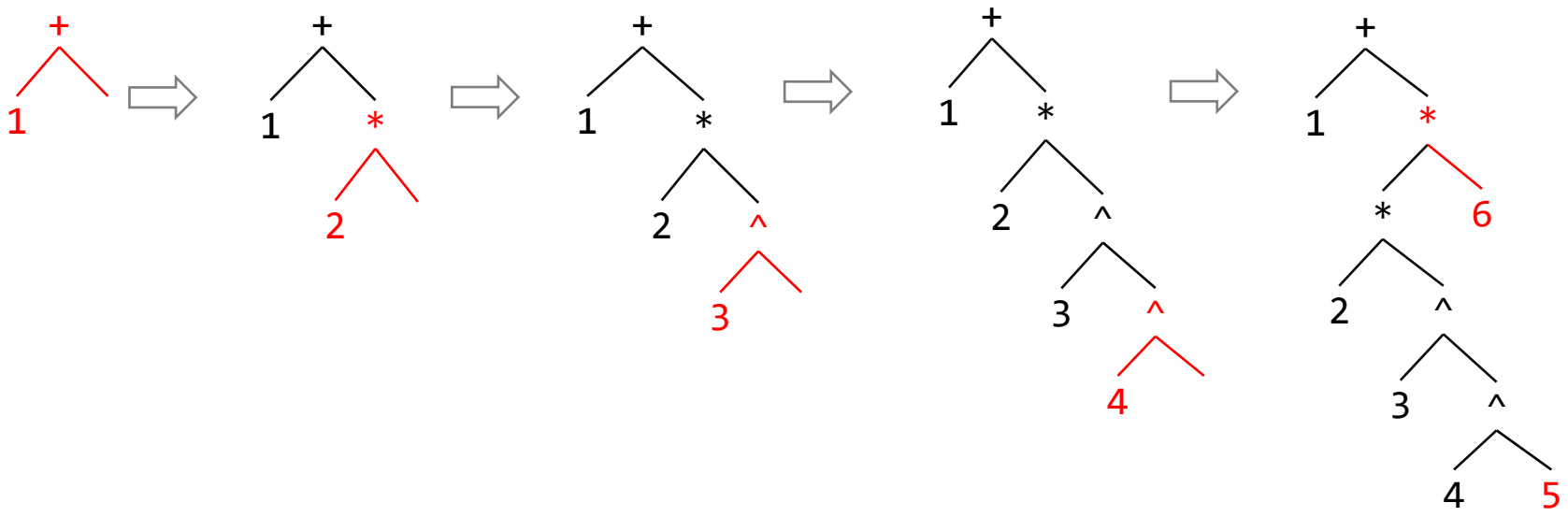


使用优先级标记

```
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

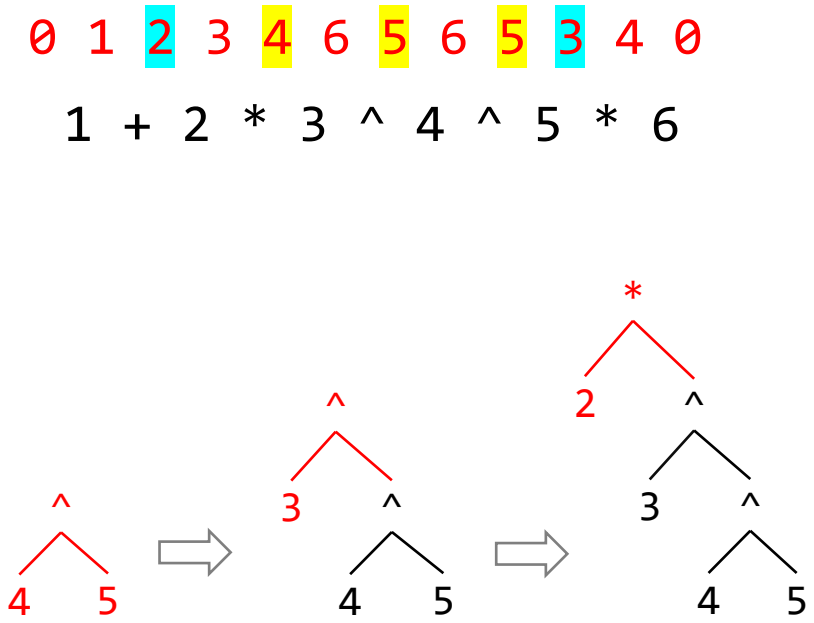
算式 1 + 2 * 3 ^ 4 ^ 5 * 6



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.type == tok::NUM
            exit -1;
        if op.type == tok::EOF
            break;
        lp, rp = Preced[op];
        if lp < precedence
            break;
        cur.next();
        right = PrattParse(cur, rp)
        left = CreateBinTree(op, left, right)
    return left;
}
```



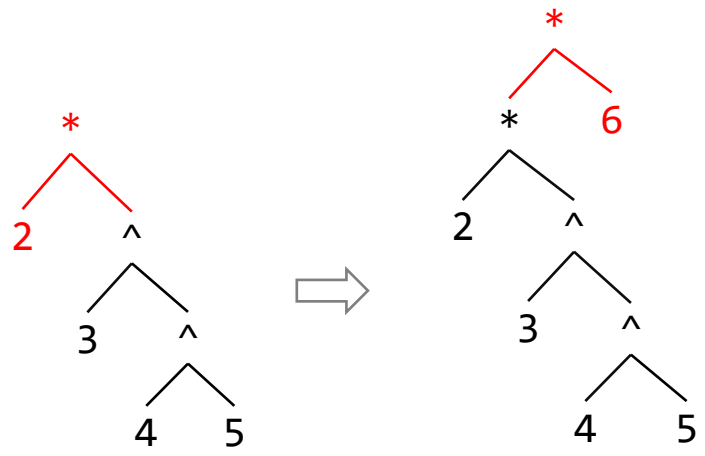
算法实现参考

```
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.type == tok::NUM
            exit -1;
        if op.type == tok::EOF
            break;
        lp, rp = Preced[op];
        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



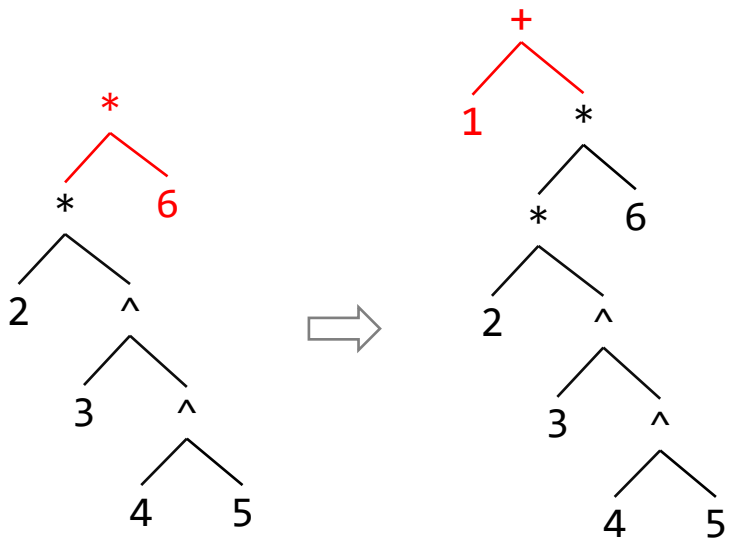
算法实现参考

```
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.type == tok::NUM
            exit -1;
        if op.type == tok::EOF
            break;
        lp, rp = Preced[op];
        if lp < precedence
            break;
        cur.next();
        right = PrattParse(cur, rp)
        left = CreateBinTree(op, left, right)
    return left;
}
```

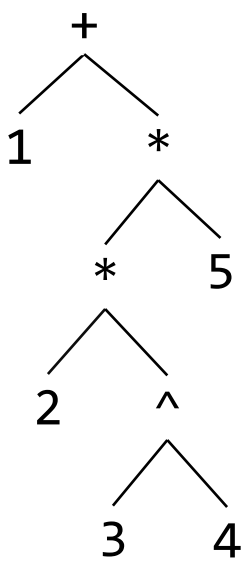
0 ~~1 2 3 4 6 5 6~~ 5 3 4 0

1 + 2 * 3 ^ 4 ^ 5 * 6

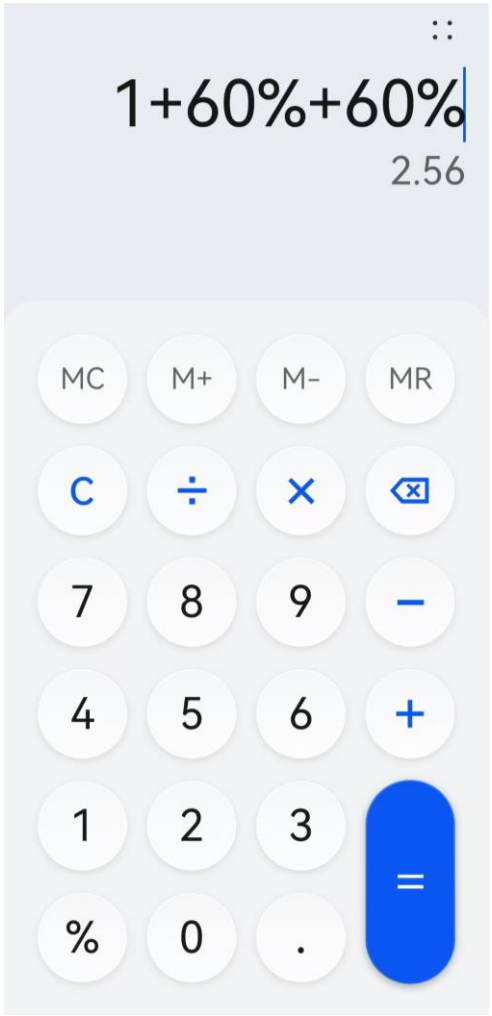


步骤3：解释执行/翻译为逆波兰表达式

- 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?

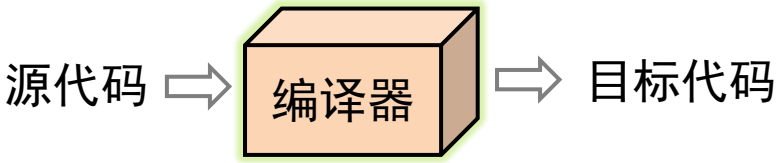


- 为什么这样?
- 如何实现的?

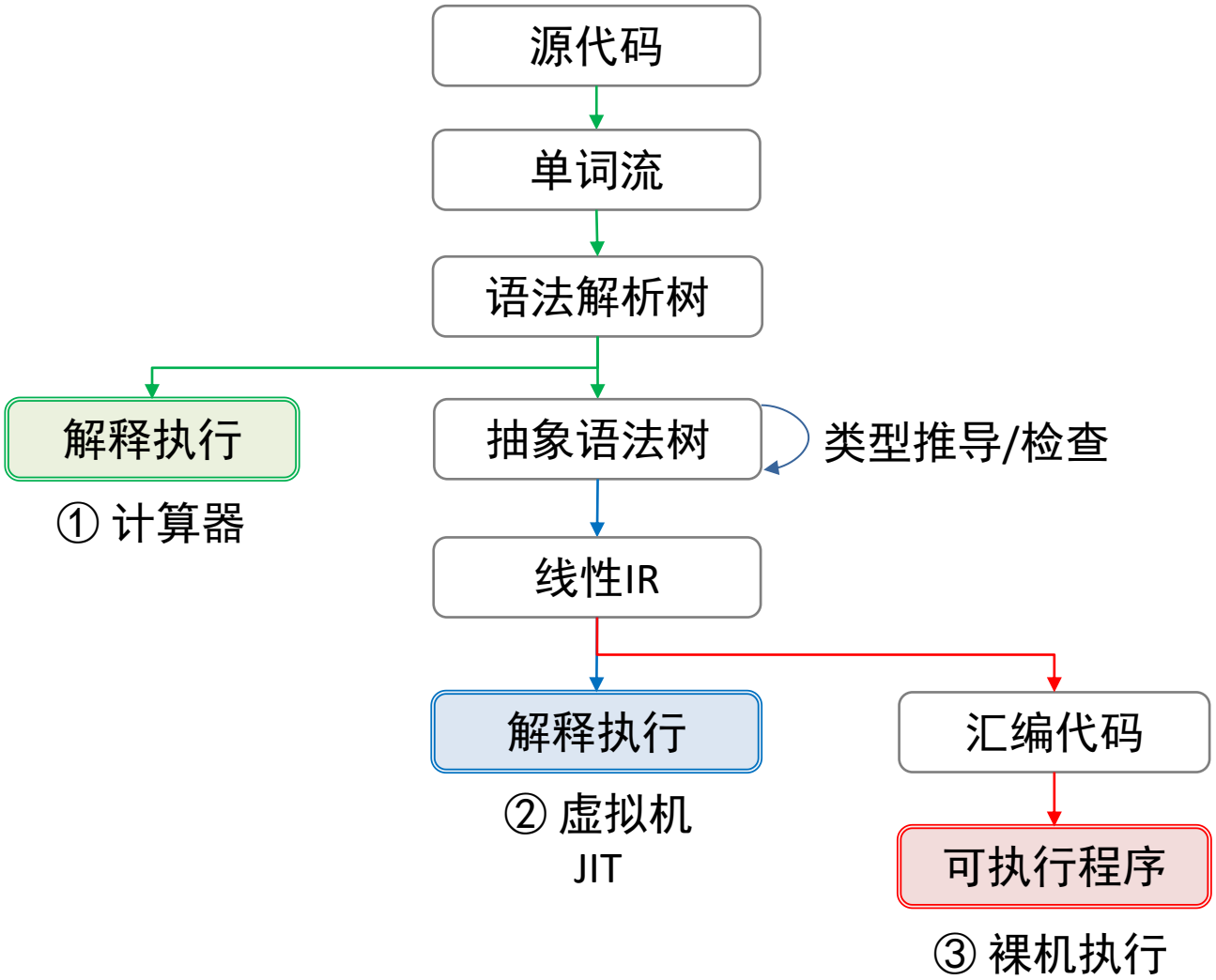
三、编译流程概览

编译

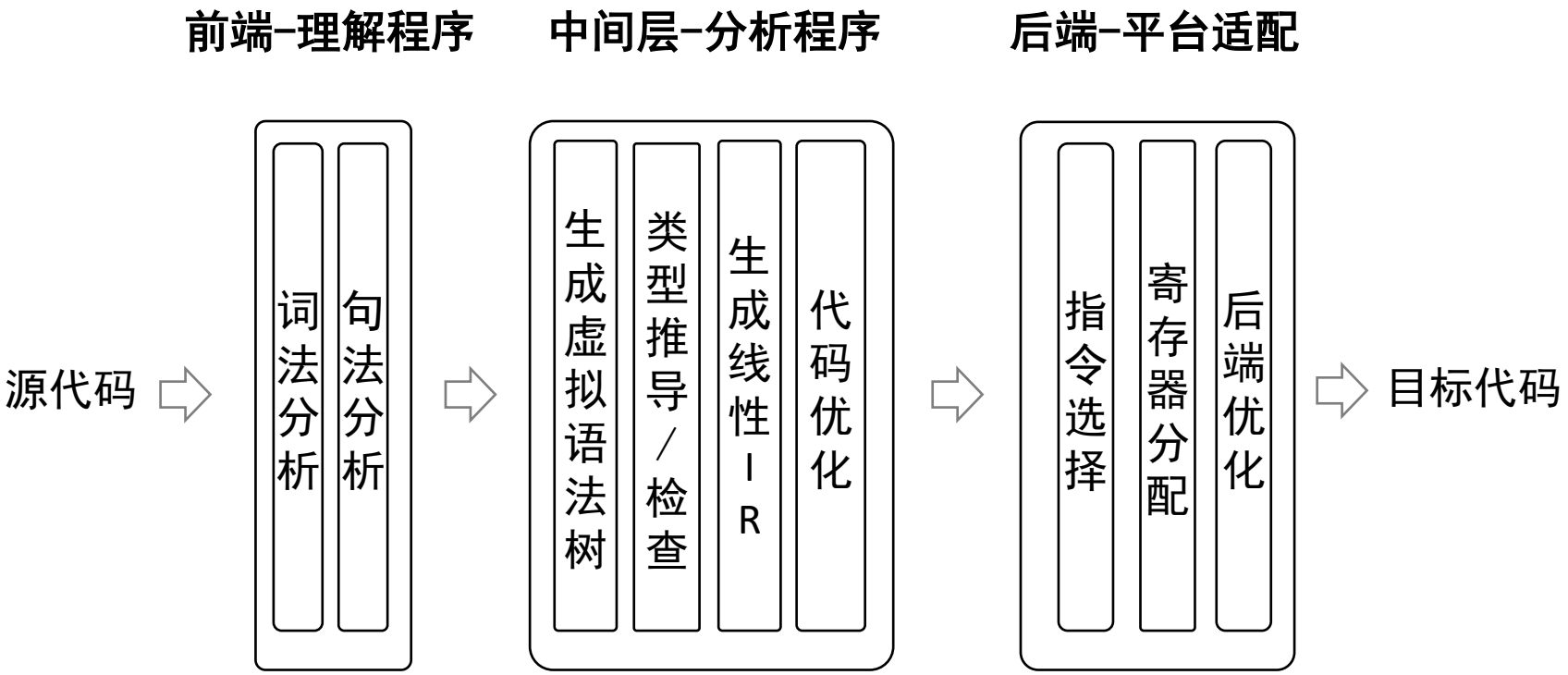
- 从一种程序语言转换为另一种程序语言
  - 源代码=>中间代码（解释执行）
    - Java/Python => Bytecode
    - C/C++/Rust => WebAssembly
  - 源代码=>汇编代码/可执行程序（编译执行）
    - C/C++/Rust => X86/Arm
- 基本要求：保持语义等价



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

- 将字符串转换为词元（token）序列
- 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

- 分析词元序列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

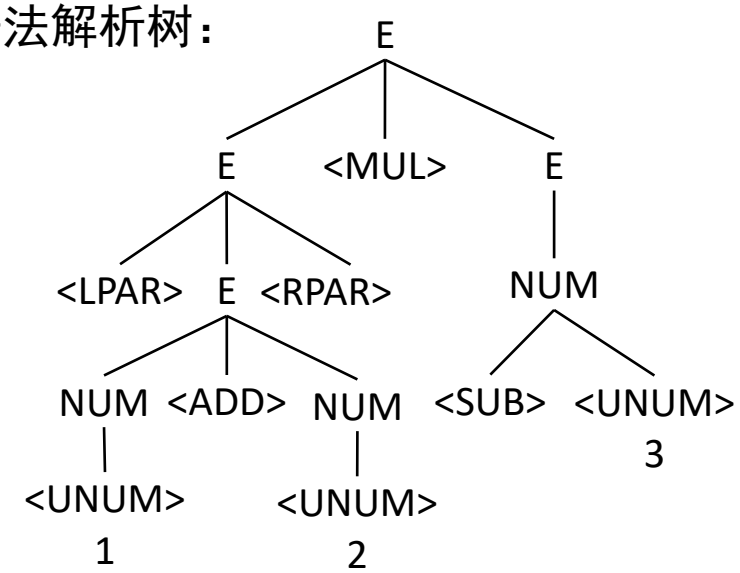
源代码：　　(1 + 2) * -3

1. 词法分析

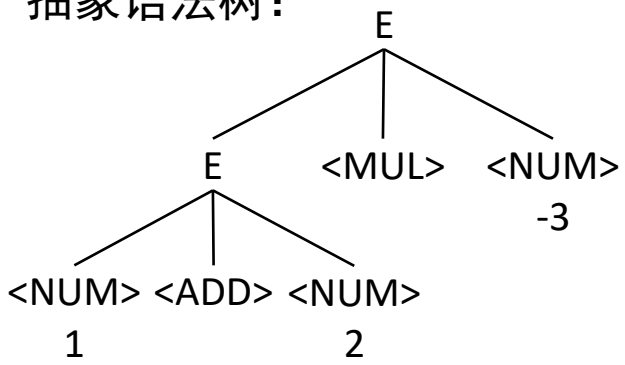
<LPAR> <UNUM(1)> <ADD> <UNUM(2)> <RPAR> <MUL> <SUB> <UNUM(3)>

2. 句法分析

语法解析树：



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

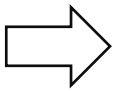
```
int t = 2 * b * c;
for (i=1; i<100; i++){
    int d = getInt();
    a = a * t * d
}
```

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
```



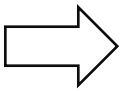
Arm汇编代码

```
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
```



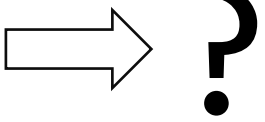
```
main:
        sub     sp, sp, #16
        mov     w8, #1
        str     w8, [sp, #28]
        str     w8, [sp, #24]
        ldr     w8, [sp, #28]
        ldr     w9, [sp, #24]
        cbnz    w8, .LBB0_2
.LBB0_1:
        add     w8, w8, w9
        str     w8, [sp, #28]
        b       .LBB0_1
.LBB0_2: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

```
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



?

通过课程学习将了解一些常见的技术概念

- 通用编程语言 vs 领域专用语言
- 静态类型 vs 动态类型
- AOT（ahead-of-time）vs 解释执行 vs JIT（just-in-time）
- 运行时环境
- ...

练习

- 实现并验证运算符优先级解析算法
  - 可以借助GPT
  - 考虑加入括号？