1 课程介绍

徐辉，xuh@fudan.edu.cn

本章学习目标：

- 了解学习编译的意义
- 了解编译流程
- 掌握运算符优先级解析算法

1.1 为什么学习编译原理？

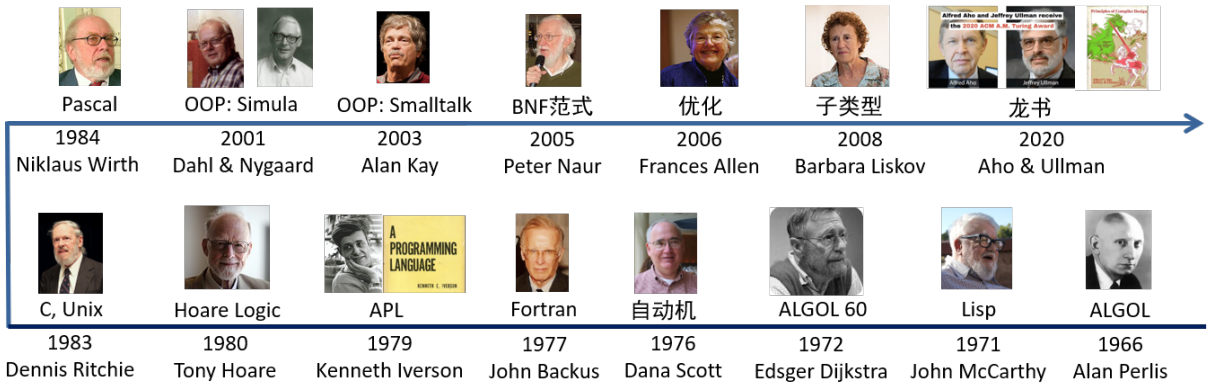

图 1.1: 与编译和语言有关的图灵奖得主

- **有用**：当旧工具不能满足新场景时，我们需要新的轮子（语言）。如图灵奖得主 Leslie Lamport 因为需要自己好用的排版工具就开发了 Latex；Mozilla 公司程序员 Graydon Hoare 为了开发安全、高效的浏览器引擎设计了 Rust 语言。近几年，随着深度学习和大模型的发展，一批与之适配的编程语言和编译技术开始涌现，如 LLVM 作者 Chris Lattner 开发的 Mojo、OpenAI 的 Triton 等。
- **经典**：历届图灵奖得主中有很多位的成就都与编译原理或编程语言有关，如图 1.1所示，有兴趣的同学们可以自己在 ACM 网站查阅[1]。

1.2 初识编译：以计算器为例

计算器可识别算式，因此可将其视为一种功能简单的编译器。本节以实现一部计算器为例分析编译器的实现思路。

1.2.1 功能需求

我们假设目标计算器如图 1.2所示，其主要功能参数如下：

- 操作数：支持整数和小数

[1] ACM 图灵奖得主：`https://amturing.acm.org/byyear.cfm`

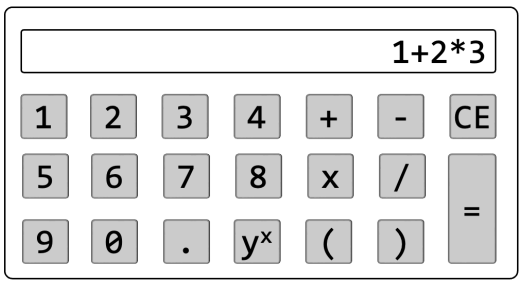

图 1.2: 目标计算器样例

- 运算符：支持加、减、乘、除四则运算和指数运算
- 括号：支持小括号

1.2.2 实现思路

要实现上述计算器，一般需要经过以下基本步骤：

1) **词法分析**：扫描算式中的操作数、运算符和括号，形成标签流。

2) **句法解析**：根据优先级、结合律等运算法则组织标签，形成语法解析树。

3) **解释执行**：根据语法解析树计算运算结果。

1.2.2.1 词法分析：识别操作数和运算符

以算式 123+456 为例，应按顺序识别出操作数 '123'、运算符 '+'、以及操作数 '456' 三个标签，并将其转换为标签流`<NUM(123)><ADD><NUM(123)>`。

算法 1 识别操作数和运算符

Input: character stream;
Output: token stream;

```
1: procedure TOKENIZE(charStream)
2:     let toks = ∅
3:     let num = ∅
4:     while ture do
5:         let cur = charStream.next();
6:         match cur :
7:             case '0'-'9' ⇒ num.append(cur); // insert at the beginning if num is empty
8:             case '+' ⇒ toks.add(num); toks.add(ADD); num.clear(); // add(num) do nothing if num is empty
9:             case '-' ⇒ toks.add(num); toks.add(SUB); num.clear();
10:            case '*' ⇒ toks.add(num); toks.add(MUL); num.clear();
11:            case '/' ⇒ toks.add(num); toks.add(DIV); num.clear();
12:            case '∧' ⇒ toks.add(num); toks.add(POW); num.clear();
13:            case '(' ⇒ toks.add(num); toks.add(LPAR); num.clear();
14:            case ')' ⇒ toks.add(num); toks.add(RPAR); num.clear();
15:            case _ ⇒ break; //EOF or an illegal character
16:        end match
17:    end while
18: end procedure
```

算法 1描述了该标签识别的思路。其关键点是使用一个缓冲区num记录当前已读取的操作数位。此步骤既不考虑算式的合法性问题（如 123 + +456），亦无需考虑 '-' 是负号还是减号的问题。

1.2.2.2 句法分析：操作符优先级解析算法

算式解析问题是一个非常经典的问题。由于我们常用的算式表示是 infix 模式，对其进行解析需要遵循优先级和结合率性质。

- 优先级（precedence）：指数运算符 > 乘除运算符 > 加减运算符
- 结合性（associativity）：加减乘除运算符均为左结合；指数运算符为右结合，如`2^3^2` = `2^(3^2)`，而非`(2^3)^2`。

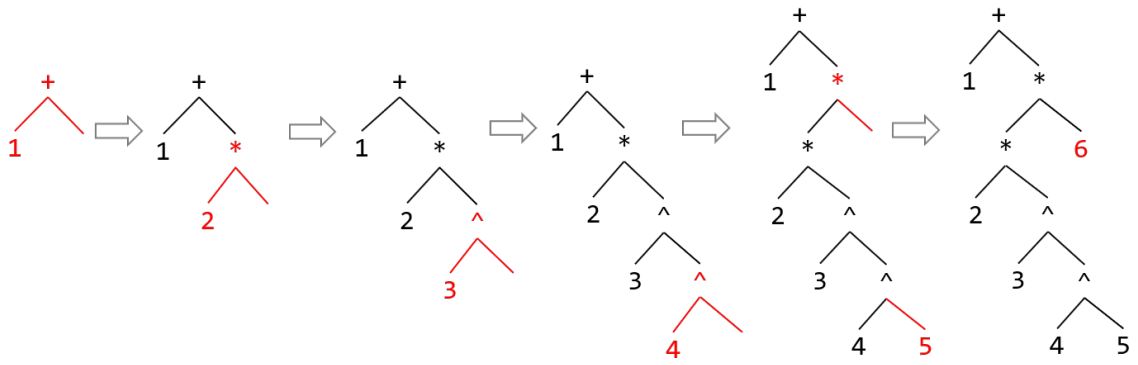

图 1.3: 算式 1+2*3^4^5*6 的解析过程

图 1.3以算式 1+2*3^4^5*6 为例阐述了无括号算式的解析过程。最终得到的解析树为满二叉树，所有叶子节点均为操作数，非叶子节点均为运算符，且每个运算符都应先于其父节点运算。该算式解析的主要思路是按照由左至右的顺序解析，并使用栈记录已经读取的运算符。具体分为以下几种情况：

- 如果当前遇到的运算符为左结合，且其优先级高于栈顶运算符的优先级，则应将该运算符作为栈顶运算符的右孩子节点。此时栈顶运算符的左孩子节点已经存在。
- 如果当前遇到的运算符为左结合，且其优先级不高于栈顶运算符的优先级，则应将其作为栈顶运算符的父节点或祖先节点，即从栈中 pop 已读取运算符直至遇到低于当前运算符优先级的运算符。
- 如果当前遇到的运算符为右结合，应将其作为栈顶运算符的右孩子节点。

优先级：	0	1	2	3	4	6	5	6	5	3	4	0
算式：		1	+	2	*	3	^	4	^	5	*	6
位置：	1	2	3	4	5	6	7	8	9	10	11	

图 1.4: 算式 1+2*3^4^5*6 的运算符优先级标注

Pratt 解析 [1] 是一种运算符优先级解析实现方法。为便于分析，该算法为每个运算符的左右两侧分别分配一个优先级数字，使其即可以体现优先级，又可以反应结合性。对于左结合的运算符，其左侧优先级低于右侧；对于右结合的运算符，则左侧优先级高于右侧。以图 1.4的优先级标注为例，运算符 '+' 和 '-' 的左右两侧优先级分别为 1 和 2，运算符 '*' 和 '/' 的左右两侧优先级分别为 3 和 4，运算符 '^' 的左右两侧优先级分别为 6 和 5。

算法 2 运算符优先级解析算法

Input: token stream, precedence (init with 0);

Output: binary parse tree;

```
1: Preced[ADD] = 1,2; Preced[SUB] = 1,2; Preced[MUL] = 3,4; Preced[DIV] = 3,4; Preced[POW] = 6,5;
2: procedure PrattParse(cur,preced)
3:     let l = cur.next(); // next() moves cur to the next position and return the value of that position.
4:     if l.type ≠ TOK::NUM then
5:         return ERROR;
6:     end if
7:     while ture do // corresponds to pop operators from the operator stack
8:         let op = cur.peek(); // peek() returns the value of the next position.
9:         match op.type :
10:            case TOK::NUM ⇒ exit ERROR;
11:            case TOK::EOF ⇒ return left;
12:        end match
13:        (lp, rp) = Preced[op];
14:        if lp < preced then:
15:            return l;
16:        end if
17:        cur.next();
18:        let r = PrattParse(cur, rp);
19:        let l = CreateBinTree(op, l, r);
20:    end while
21:    return l;
22: end procedure
```

算法 2给出了 Pratt 算法的伪代码实现。令初始位置优先级为 0，调用 PrattParse 函数即可得到图 1.3中的语法解析树。过程如下：

表 1.1: PrattParse 解析过程

cur	preced	l	op	lp	rp	操作
0	0	1	+	1	2	r = PrattParse(cur, rp); l = CreateBinTree(peek, l, r);
2	2	2	*	3	4	r = PrattParse(cur, rp); l = CreateBinTree(peek, l, r);
4	4	3	^	5	6	r = PrattParse(cur, rp); l = CreateBinTree(peek, l, r);
6	6	4	^	5	6	r = PrattParse(cur, rp); l = CreateBinTree(peek, l, r);
8	6	5	*	3	4	return l;
8	6	^(4,5)	*	3	4	return l;
8	4	^(3,^(4,5))	*	3	4	return l;
8	2	*(2,^(3,^(4,5)))	*	3	4	r = PrattParse(cur, rp); l = CreateBinTree(peek, l, r);
10	4	6	EOF	-	-	return l;
10	2	*(*(2,^(3,^(4,5)),6)	EOF	-	-	return l;
10	0	+(1,*(*(2,^(3,^(4,5)),6))	EOF	-	-	return l;

1.2.2.3 解释执行：逆波兰表达式

基于语法解析树，便可对其进行后续遍历完成算式计算。对于计算器程序来说，我们也可以先将其转化为逆波兰表达式（Reverse Polish Notation），即对语法解析树进行后序遍历得到的符号序列，如`1+2*3^4^5*6`的逆波兰表达式是：`1 2 3 4 5 ^ ^ * 6 * +`。逆波兰表达式非常易于计算：按照顺序读取字符串，如果遇到操作数则入栈；如果遇到运算符，则弹出栈顶的两个操作数，求值后将结果入栈。字

符串读取完毕后，栈顶元素就是最终结果。

1.3 编译流程概览

由于编程语言比算式复杂，真实的编译器要比计算器复杂的多。图 1.5展示了编译的主要流程和技术分支。由于算式复杂度低，可直接被解释执行。而一般的通用编程语言都是图灵完备的，因此用其编写的程序都需要通用图灵机来运行，在实际应用时体现为虚拟机和实机两种方式。本学期后面的课程会对上述过程进行详细讲解。

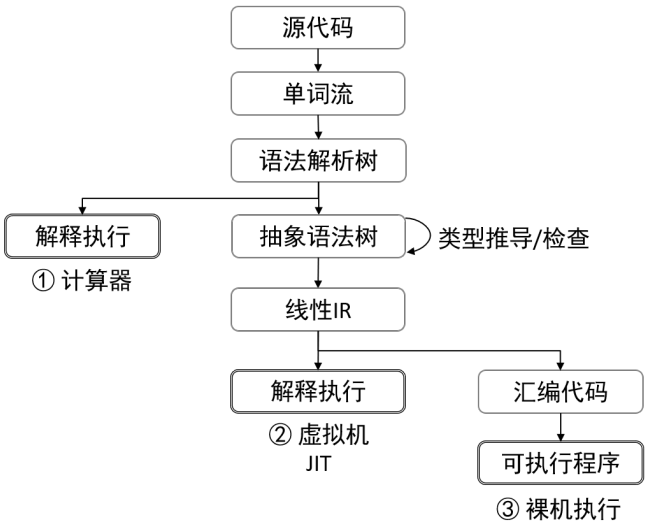

图 1.5: 编译流程

练习

1. 一些计算器输入算式：1+60%+60%，计算结果为 2.56，请分析其实现原理。
2. 实现 pratt 算法并验证其正确性：i. 不考虑括号；ii. 考虑括号。
3. 你日常学习和工作中用到的哪些技术或工具与编译有关？举例说明。

Bibliography

[1] Vaughan R. Pratt, “Top down operator precedence.” *In Proceedings of the 1st annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*, 1973.