1 课程介绍

徐辉，xuh@fudan.edu.cn

本章学习目标：

- 了解学习编译原理的意义。
- 掌握运算符优先级解析算法。
- 初识编译流程。

1.1 为什么学习编译原理？

本材料为理工科教学使用，因此会省略非必要的文字赘述。

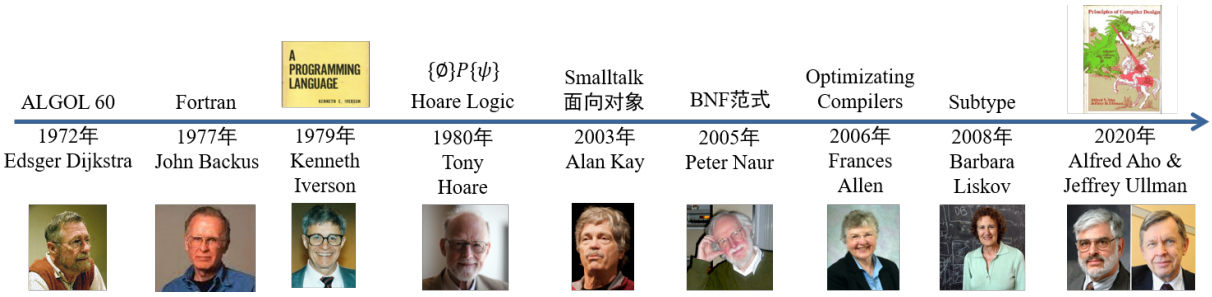

图 1.1: 与编译和语言有关的图灵奖得主

- 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。
- 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

1.2.1 功能需求

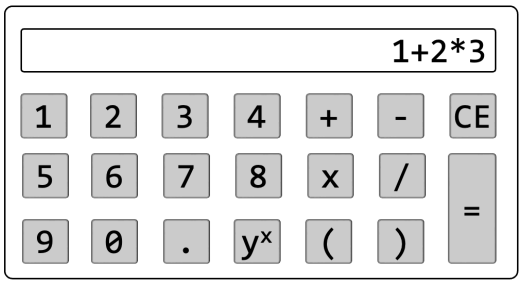

图 1.2: 目标计算器样例

我们假设计算器的功能描述如下：

- 支持整数和小数。
- 支持加、减、乘、除四则运算和指数运算。
- 支持括号。

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：扫描用户输入字符流中的操作数和运算符，将其转化为相应的标签流。

2) 句法解析：将标签流按照优先级、结合律、以及括号等运算规则进行组织，形成语法解析树。

3) 语法制导：将语法解析树翻译成实际的计算代码。

1.2.2.1 词法分析：识别算数和运算符

以算式 123+456 为例，应按顺序识别出 <NUM(123)>、<ADD>、<NUM(123)> 三个标签。算法 1描述了基本的词法分析思路。此步骤即不考虑算式的合法性问题（如 123++456），又不考虑 '-' 是负号还是减号的问题。

算法 1 识别算数和运算符

```
Input: character stream;
Output: token stream;
1: procedure TOKENIZE(charStream)
2:     let num = ∅
3:     while ture do
4:         let cur = charStream.next();
5:         match cur :
6:             case '0'-'9' ⇒  // a digit is a part of a number
7:                 num.append(cur); // if num is empty, append() inserts the digit at the begining
8:             case '+' ⇒  // an operator
9:                 tok.add(num); tok.add(ADD); num.clear(); // add() and empty() do nothing if num is empty;
10:            case '-' ⇒  // an operator or a negative number?
11:                tok.add(num); tok.add(SUB); num.clear();
12:            case '*' ⇒
13:                tok.add(num); tok.add(MUL); num.clear();
14:            case '/' ⇒
15:                tok.add(num); tok.add(DIV); num.clear();
16:            case '∧' ⇒
17:                tok.add(num); tok.add(POW); num.clear();
18:            case '(' ⇒
19:                tok.add(num); tok.add(LPAR); num.clear();
20:            case ')' ⇒
21:                tok.add(num); tok.add(RPAR); num.clear();
22:            case _ ⇒
23:                break; //EOF or illegal character;
24:        end match
25:    end while
26: end procedure
```

1.2.2.2 句法解析：操作符优先级解析算法

算式解析问题是一个非常经典的解析问题，我们常用的算式是 infix 模式，对其进行解析需要考虑其中的优先级和结合性信息。

- 优先级（precedence）：指数运算优先级 > 乘除运算 > 加减运算
- 结合性（associativity）：加减乘除运算为左结合，指数运算为右结合，如 2^3 ^2 = 2^(3^2) ≠ (2^3)^2。

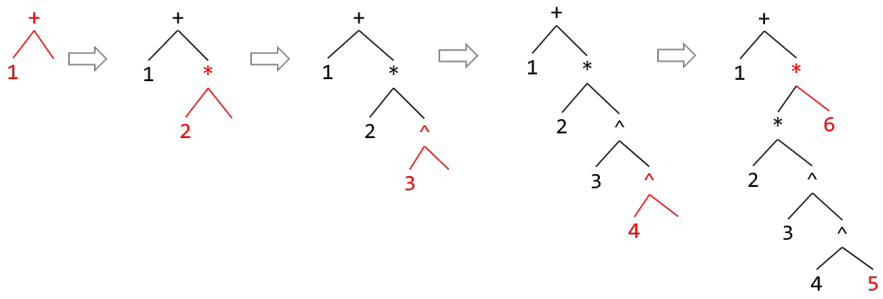

图 1.3: 算式 1+2*3^4^5+6 的解析过程

图 1.3以算式 1+2*3^4^5+6 的解析为例阐述了无括号算式的解析过程。其本质是按照从左至右的顺序解析，具体分为以下几种情况：

- 如果当前遇到的运算符为左结合，且当前运算符优先级 > 栈顶运算符的优先级，则当前运算符应作为栈顶运算符的右孩子节点。注：1）使用栈记录已经解析的运算符；2）该解析树最终应为满二叉树，且当前运算符的左孩子节点已经存在，因此放在右孩子节点；3）因此如果一个运算符的子树为满二叉树，应从栈中 pop。
- 如果当前遇到的运算符为左结合，且当前运算符优先级 ≤ 栈顶运算符的优先级，则当前运算符应作为栈顶运算符的父节点或祖先节点。注：pop 直到当前运算符优先级 > 栈顶运算符优先级。
- 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。

Pratt Parsing [2] 是一种巧妙实现上述思路的方法。算法 2给出了一种参考实现方式，该算法通过为每个运算符的左侧和右侧赋上优先级（算式两侧优先级为 0），巧妙的实现了上述解析过程。

上述算法可以扩展到支持括号，即将括号内容其当作一棵子树重复上述过程即可。

1.2.2.3 语法制导：逆波兰表达式

有了语法解析树，便可以对树进行后续遍历完成计算。对于计算器程序来说，我们可以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。

1) 按照顺序读取字符串，如果遇到操作数则入栈；
2) 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；
3) 字符串扫描一遍后，栈顶就是表达式的值。

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
2: Preced[SUB] = 1,2
3: Preced[MUL] = 3,4
4: Preced[DIV] = 3,4
5: Preced[POW] = 6,5
6: procedure PrattParse(cur,preced)BinTree
7:     let root = ∅;
8:     let elem = cur.next();
9:     if left.type ≠ TOK::UNUM then
10:        return ERROR;
11:    end if
12:    while ture do
13:        let peek = cur.peek();
14:        if peek.type ≠ TOK::BINOP then:
15:            return ERROR;
16:        end if
17:        (lp, rp) = Preced[op];
18:        if lp < preced then:
19:            break;
20:        end if
21:        cur.next();
22:        let right = PrattParse(cur, rp);
23:        let root = CreateBinTree(peek, root, right);
24:    end while
25:    return root;
26: end procedure
```

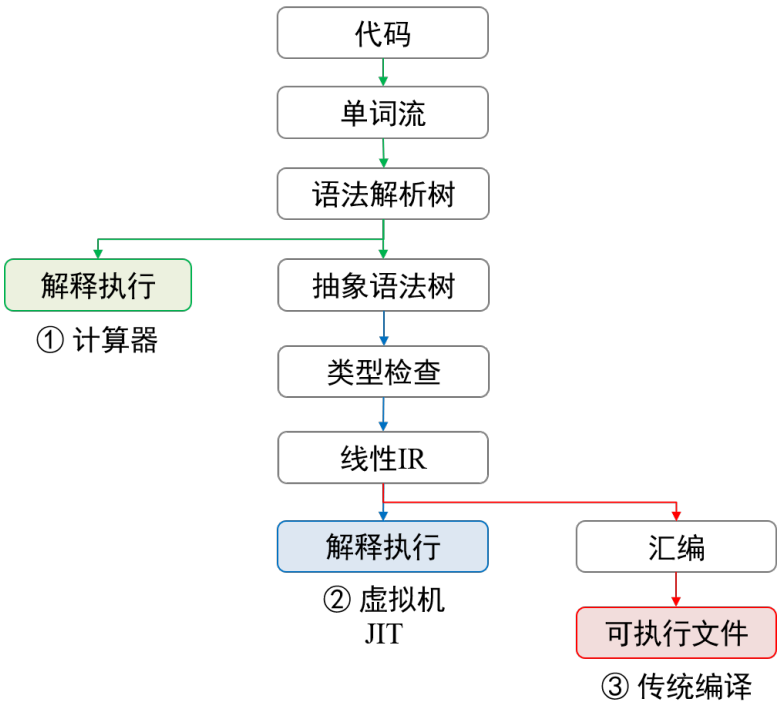

图 1.4: 编程流程

1.3 编译流程概览

图 1.4展示了编译的主要技术范畴和分支。计算器是其中最简单的一种形式，由于算式复杂度低，无需虚拟机即可直接解释执行。而一般的采用通用编程语言编写的程序都需要图灵机来运行，具体分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。

练习

1. 实现并验证 pratt 算法（i. 不考虑括号；ii. 考虑括号）。

Bibliography

[1] Turing Award, `https://amturing.acm.org/byyear.cfm`.

[2] Vaughan R. Pratt, "Top down operator precedence." In Proceedings of the 1st annual ACM SIGACT-SIGPLAN symposium on Principles of programming languages, 1973.