# 1. 编程基础1：编程语言概述

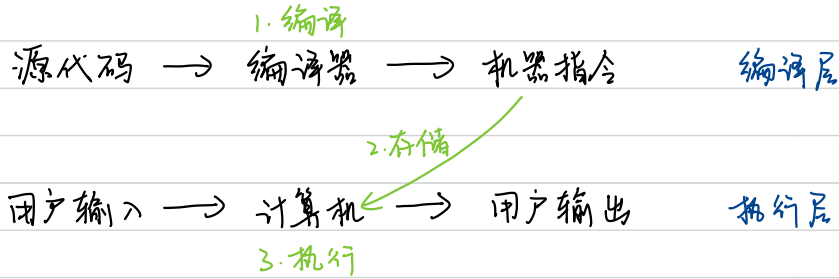
## 一. 计算机硬件，编译和执行.

### 1. 冯诺依曼结构

(1) 机器指令存储在内存里面.

(2) 机器指令在ALU中执行.

### 2. 编译和执行




三种编译执行

(1) 编译、执行(C)

(2) 直接解释执行(Basic)

(3) 编译成字节码、解释执行(Java)

### 3. 编译与解释

编译(Compile)：把整个程序源代码翻译成另外一种代码，然后等待被执行，发生在运行之前，产物是另一份代码.

解释(Interpret)：把程序源代码逐行读懂然后执行，发生在运行时，产物是运行结果.

## 二. 编译器的结构.

- 词法分析
- 语法分析
- 语义分析
- 中间代码生成

(以上) 机器无关

- 代码优化
- 目标代码生成

(以上) 机器相关

(1) 词法分析 (Lexical Analysis)

扫描输入的符号，然后将这些符号分组形成具有一定含义的单元.

Example:

输入: temp := (x+)

输出: temp := ( x + )

(2) 语法分析 (Syntactic Analysis)

单词符号被分组成为语义单元，输出解析树.

Example:

输入: result := salary + bonus * 1.10

输出:

Example:

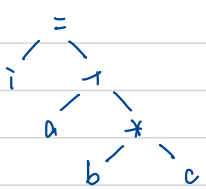
(源代码) i := a + b * c

↓ 词法分析

(单词流) i = a + b * c

↓ 语法分析

(语法树)




(3) 语义分析 (Semantic Analysis)

语义检查：temp和x的类型是由它们的声明决定的，不同的声明语义是不一样的.

- * 可以是整数乘法，也可以是浮点数乘法

输出一个打展的解析树，表示程序的句法结构，同时还包含了标示符的类型、标示符在哪里声明等信息.

## 14) 中间代码生成

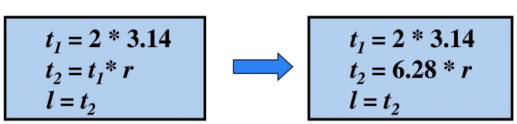
| 序号 | 指令类型 | 指令形式 |
|---|---|---|
| 1 | 赋值指令 | $x = y$ op $z$<br>$x =$ op $y$ |
| 2 | 复制指令 | $x = y$ |
| 3 | 条件跳转 | if $x$ relop $y$ goto $n$ |
| 4 | 非条件跳转 | goto $n$ |
| 5 | 参数传递 | param $x$ |
| 6 | 过程调用 | call $p$, $n$ |
| 7 | 过程返回 | return $x$ |
| 8 | 数组引用 | $x = y[i]$ |
| 9 | 数组赋值 | $x[i] = y$ |
| 10 | 地址及指针操作 | $x = \&y$<br>$x = *y$<br>$*x = y$ |

- $x = y$ op $z$ ( op , $y$ , $z$ , $x$ )
- $x =$ op $y$ ( op , $y$ , _ , $x$ )
- $x = y$ ( = , $y$ , _ , $x$ )
- if $x$ relop $y$ goto $n$ ( relop , $x$ , $y$ , $n$ )
- goto $n$ ( goto , _ , _ , $n$ )
- param $x$ (param, _ , _ , $x$ )
- call $p$, $n$ ( call , $p$ , $n$ , _ )
- return $x$ (return, _ , _ , $x$ )
- $x = y[i]$ ( =[] , $y$ , $i$ , $x$ )
- $x[i] = y$ ( []= , $y$ , $x$ , $i$ )
- $x = \&y$ ( & , $y$ , _ , $x$ )
- $x = *y$ ( =* , $y$ , _ , $x$ )
- $*x = y$ ( *= , $y$ , _ , $x$ )

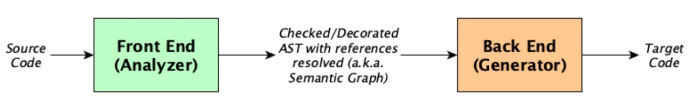
## 15) 代码优化

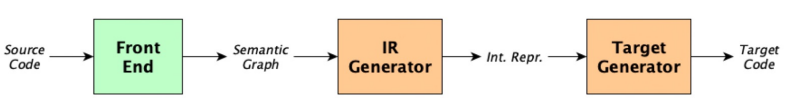
- 删除公共子表达式
- 代码外提
- 强度削弱
- 变换循环控制条件
- 合并已知变量
- 复写传播
- 删除无用代码
- 常量合并
- 代码移动
- 内联函数的调用

常量合并:




## 16) 目标代码生成



Example.

```
int foo(int a, int b){
    return a + b + 10;
}
```

执行"clang -S foo.c -o foo.x86.s”命令，可以得到对应的X86架构下的汇编

```
#序曲
pushq  %rbp
movq  %rsp, %rbp      #%rbp是栈底指针

#函数体
movl  %edi, -4(%rbp)  #把第1个参数写到栈里第一个位置（偏移量为4）
movl  %esi, -8(%rbp)  #把第2个参数写到栈里第二个位置（偏移量为8）
movl  -4(%rbp), %eax  #把第1个参数写到%eax寄存器
addl  -8(%rbp), %eax  #把第2个参数加到%eax
addl  $10, %eax       #把立即数10加到%eax, %eax同时是放返回值的地方

#尾声
popq  %rbp
retq
```

```
int foo(int a, int b){
    return a + b + 10;
}
```

使用"clang -S -target armv7a-none-eabi foo.c -o foo.armv7a.s"命令，生成一段针对ARM芯片的汇编代码：

```
//序曲
sub sp, sp, #8     //把栈扩展8个字节，用于放两个参数，sp是栈顶指针

//函数体
str r0, [sp, #4]   //把第1个参数写到栈顶+4的位置
str r1, [sp]       //把第2个参数写到栈顶位置
ldr r0, [sp, #4]   //把第1个参数从栈里加载到r0寄存器
ldr r1, [sp]       //把第2个参数从站立加载到r1寄存器
add r0, r0, r1     //把r1加到r0,结果保存在r0
add r0, r0, #10    //把常量10加载到r0，结果保存在r0,r0也是放返回值的地方

//尾声
add sp, sp, #8     //缩减栈
bx  lr             //返回
```

## 三. 代码和语言.

### 第一代编程语言:

- 1951 – Regional Assembly Language
- 1952 – Autocode
- 1954 – IPL (forerunner to LISP)
- 1955 – FLOW-MATIC (led to COBOL)
- 1957 – FORTRAN (First compiler)
- 1957 – COMTRAN (precursor to COBOL)
- 1958 – LISP
- 1958 – ALGOL 58
- 1959 – FACT (forerunner to COBOL)
- 1959 – COBOL
- 1959 – RPG
- 1962 – APL
- 1962 – Simula
- 1962 – SNOBOL
- 1963 – CPL (forerunner to C)
- 1964 – Speakeasy (computational environment)
- 1964 – BASIC
- 1964 – PL/I
- 1966 – JOSS
- 1967 – BCPL (forerunner to C)

### 基础范式建立:

- 1968 – Logo
- 1969 – B (forerunner to C)
- 1970 – Pascal
- 1970 – Forth
- 1972 – C（结构化编程范式）
- 1972 – Smalltalk（面向对象编程范式）
- 1972 – Prolog（逻辑编程范式）
- 1973 – ML
- 1975 – Scheme（函数式编程范式）
- 1978 – SQL (a query language, later extended)（DSL）

## 模块化编程（80年代）

- 1980 – C++ (as C with classes, renamed in 1983)
- 1983 – Ada
- 1984 – Common Lisp
- 1984 – MATLAB
- 1985 – Eiffel
- 1986 – Objective-C
- 1986 – Erlang
- 1987 – Perl
- 1988 – Tcl
- 1988 – Wolfram Language (as part of Mathematica, only got a separate name in June 2013)
- 1989 – FL (Backus)

## 互联网时代（90年代）

- 1990 – Haskell
- 1991 – Python
- 1991 – Visual Basic
- 1993 – Ruby
- 1993 – Lua
- 1993 – R
- 1994 – CLOS (part of ANSI Common Lisp)
- 1995 – Ada 95
- 1995 – Java
- 1995 – Delphi (Object Pascal)
- 1995 – JavaScript
- 1995 – PHP
- 1997 – Rebol
- 1999 – D

现趋势：

- 2000 – ActionScript
- 2001 – C#
- 2003 – Apache Groovy
- 2003 – Scala
- 2005 – F#
- 2006 – Windows PowerShell
- 2007 – Clojure
- 2009 – Go
- 2010 – Rust
- 2011 – Dart
- 2012 – Julia
- 2014 – Swift