中国科学技术大学
University of Science and Technology of China

引　论

《编译原理和技术》

张昱

0551-63603804，yuzhang@ustc.edu.cn

中国科学技术大学

计算机科学与技术学院

主要内容

1. 编程语言及设计
2. 编译器及形式
3. 编译器的阶段
4. 示例：程序的表示
5. 基础实验的考虑

主要内容

1. 编程语言及设计
2. 编译器及形式
3. 编译器的阶段
4. 示例：程序的表示
5. 基础实验的考虑

编程语言

- 什么是编程语言
  - **A programming language** is a notation for describing computations to people and to machines.
- 每种编程语言有自己的计算模型
  - 过程型(Procedural): C, C++, C#, Java
  - 声明型(Declarative): SQL, …
  - 逻辑型(Logic): Prolog, …
  - 函数式(Functional): Lisp/Scheme, Haskell, ML, …
  - 脚本型(Scripting): AWK, Perl, Python, PHP, Ruby, …

求最大公约数 gcd

```
int gcd(int a, int b) {                                  // C
    while (a != b) {
        if (a > b) a = a - b;
        else b = b - a;
    }
    return a;
}

let rec gcd a b =                                        (* OCaml *)
    if a = b then a
    else if a > b then gcd b (a - b)
         else gcd a (b - a)

gcd(A,B,G) :- A = B, G = A.                              % Prolog
gcd(A,B,G) :- A > B, C is A-B, gcd(C,B,G).
gcd(A,B,G) :- B > A, C is B-A, gcd(C,A,G).
```

编程语言百花齐放

https://octoverse.github.com/

- 众多：GitHub -- 开源项目涉及316种编程语言
  - http://www.99-bottles-of-beer.net/ 用1500种语言编写
- 语言自身在不断发展
  - C - C90, C99, C11； C++ - 1998, 2003, 2006, 2011, 2014
- 最流行的语言(动态变化)

GitHub(2016)：1JavaScript, 2Java, 3Python, 4Ruby, 5PHP, 6C

Tiobe https://www.tiobe.com/tiobe-index/

- 2017.8：1Java, 2C, 3C++, 4C#, 5Python, 6VB .NET
- 2010年：1Java, 2C, 3PHP, 4C++, 5VB, 6C#, 7Python
- 1970年：1Fortran, 2Lisp, 3Cobol, 4Algol60, 5APL, 6Snobol4

99 Bottles of Beer

99 bottles of beer on the wall, 99 bottles of beer.
Take one down and pass it around, 98 bottles of beer on the wall.

98 bottles of beer on the wall, 98 bottles of beer.
Take one down and pass it around, 97 bottles of beer on the wall.
.
.
.
2 bottles of beer on the wall, 2 bottles of beer.
Take one down and pass it around, 1 bottle of beer on the wall.

1 bottle of beer on the wall, 1 bottle of beer.
Take one down and pass it around, no more bottles of beer on the wall.

No more bottles of beer on the wall, no more bottles of beer.
Go to the store and buy some more, 99 bottles of beer on the wall.

[Traditional]

C: 99 Bottles of Beer

```
#define MAXBEER (99)
void chug(int beers);
main() {
  register beers;
  for(beers = MAXBEER; beers; chug(beers--))
    puts("");

  puts("\nTime to buy more beer!\n");
  exit(0);
}
void chug(register beers) {
  char howmany[8], *s;
  s = beers != 1 ? "s" : "";
  printf("%d bottle%s of beer on the wall,\n", beers, s);
  printf("%d bottle%s of beeeeer . . . ,\n", beers, s);
  printf("Take one down, pass it around,\n");

  if(--beers) sprintf(howmany, "%d", beers); else strcpy(howmany, "No more");
  s = beers != 1 ? "s" : "";
  printf("%s bottle%s of beer on the wall.\n", howmany, s);
}
```

[Bill Wein]

Java: 99 Bottles of Beer

```
class bottles {

  public static void main(String args[]) {
    String s = "s";

    for (int beers=99; beers>-1;) {
      System.out.print(beers + " bottle" + s + " of beer on the wall, ");
      System.out.println(beers + " bottle" + s + " of beer, ");

      if (beers==0) {
        System.out.print("Go to the store, buy some more, ");
        System.out.println("99 bottles of beer on the wall.\n");
        System.exit(0);
      } else
        System.out.print("Take one down, pass it around, ");

      s = (--beers == 1)?"":"s";
      System.out.println(beers + " bottle" + s + " of beer on the wall.\n");
    }
  }
}
```

[Sean Russell]

AWK: 99 Bottles of Beer

```
BEGIN {
  for(i = 99; i >= 0; i--) {
    print ubottle(i), "on the wall,", lbottle(i) "."
    print action(i), lbottle(inext(i)), "on the wall."
    print
  }
}
function ubottle(n) {
 return sprintf("%s bottle%s of beer", n ? n : "No more", n - 1 ? "s" : "")
}
function lbottle(n) {
 return sprintf("%s bottle%s of beer", n ? n : "no more", n - 1 ? "s" : "")
}
function action(n) {
  return sprintf("%s", n ? "Take one down and pass it around," : \
                         "Go to the store and buy some more,")
}
function inext(n) {
  return n ? n - 1 : 99
}
```

[Osamu Aoki, http://people.debian.org/~osamu]

程序语言的设计

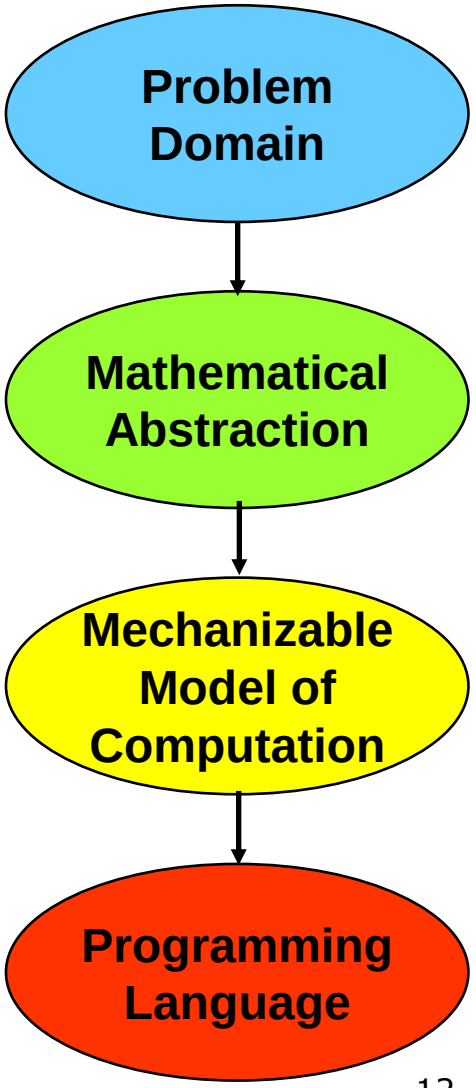
- 为什么那么多语言？
  - 单个语言不能适用所有应用
  - 程序员对语言的好坏、如何编程有自己的观点和看法
  - 没有评价语言好坏的普遍接受的标准
- 语言进化之驱动力
  - 应用的多样性
  - 提高软件开发生产力(productivity)
  - 改善软件的安全性、可靠性和可维护性
  - 支持并行(parallelism)与并发(concurrency)
  - 移动和分发，模块化，多范型

程序语言设计的计算思维

计算思维 (Computational Thinking)

Jeannette M. Wing
Computational Thinking
CACM, vol. 49, no. 3, pp. 33-35, 2006

Computational thinking is a **fundamental skill for everyone**, not just for computer scientists. To reading, writing, and arithmetic, **we should add computational thinking to every child's analytical ability**. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking.

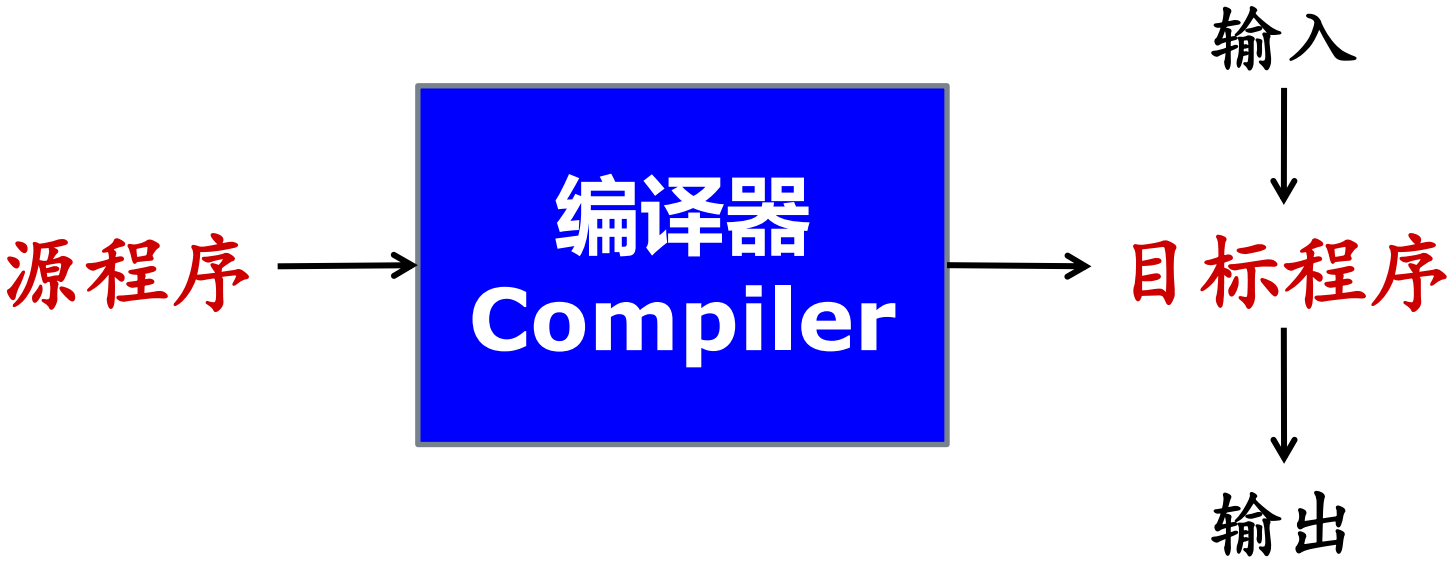


语言设计中的计算思维

主要内容

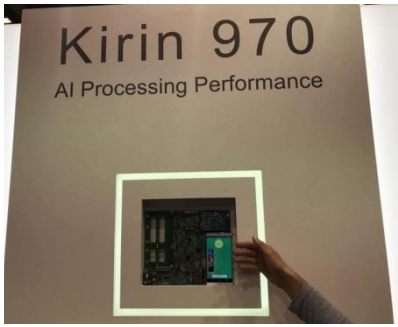
1. 编程语言及设计
2. **编译器及形式**
3. 编译器的阶段
4. 示例：程序的表示
5. 基础实验的考虑

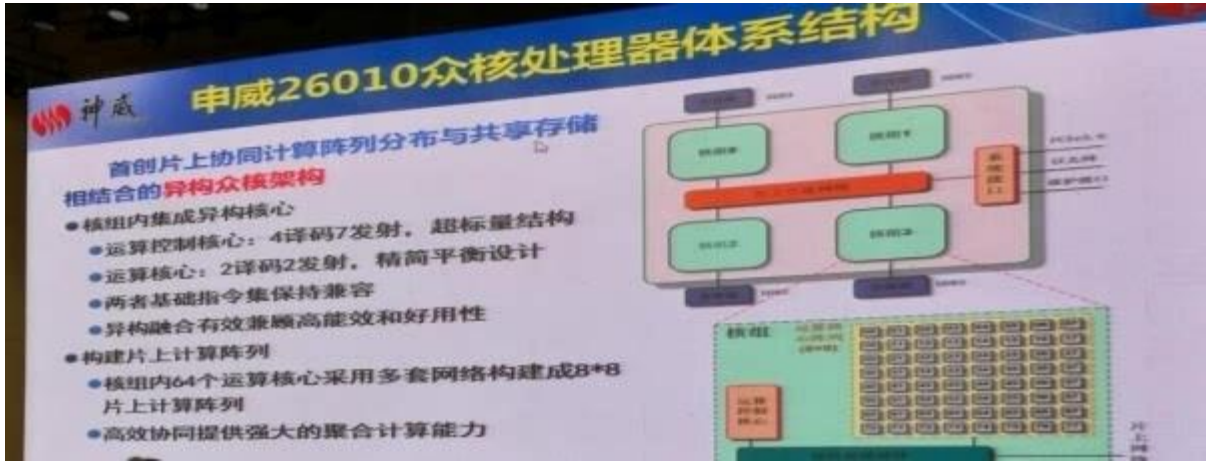
编译器是什么



目标语言

- 另一种编程语言
- CISCs（复杂指令集）：x86、IA64、…
- RISCs（精简指令集）：MIPS、ARM、…
- 多核/众核
- GPUs：CUDA、OpenCL
- FPGAs
- 量子计算机
- TPU，NPU
- …

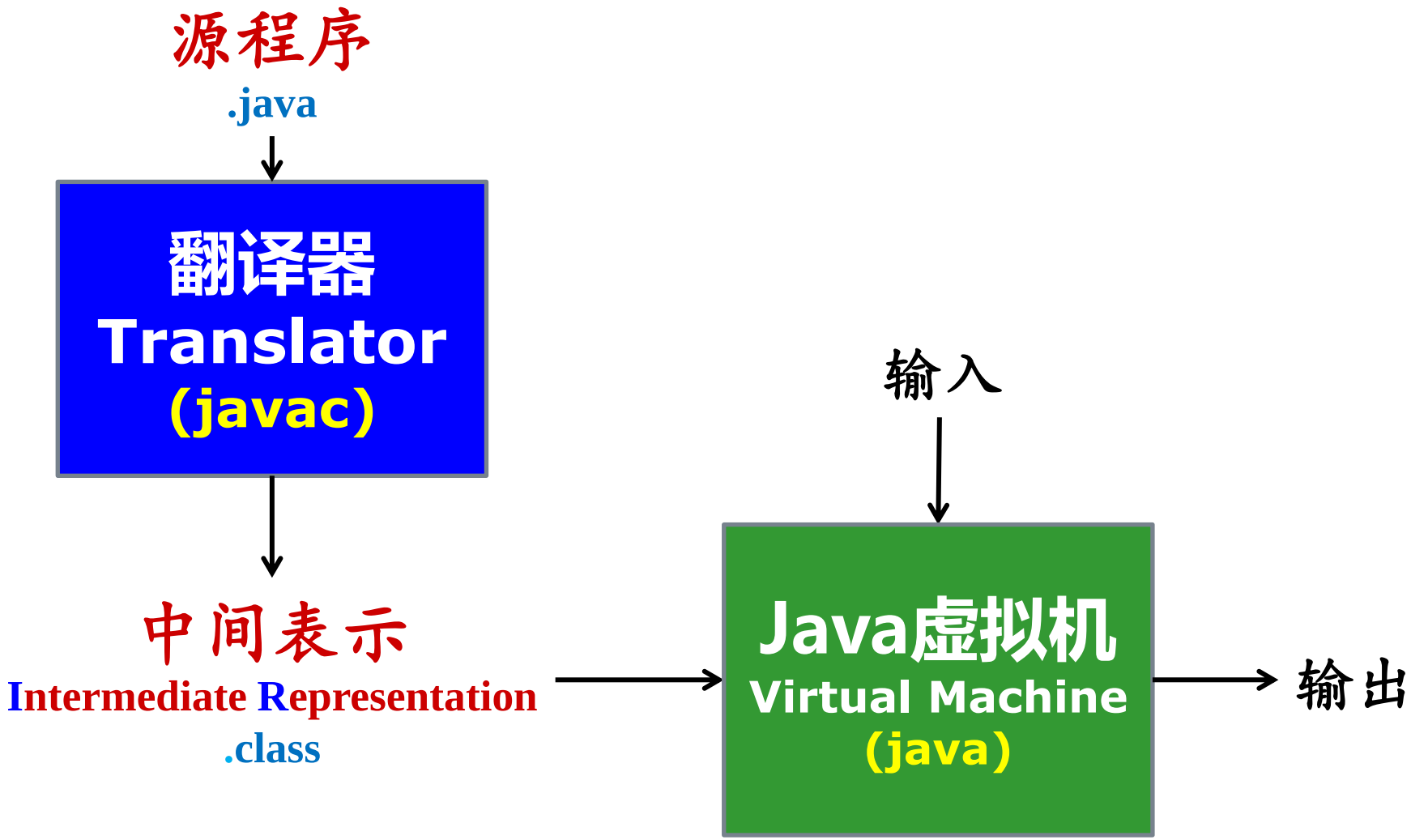




解释器

直接在输入上执行源程序

Java编译器

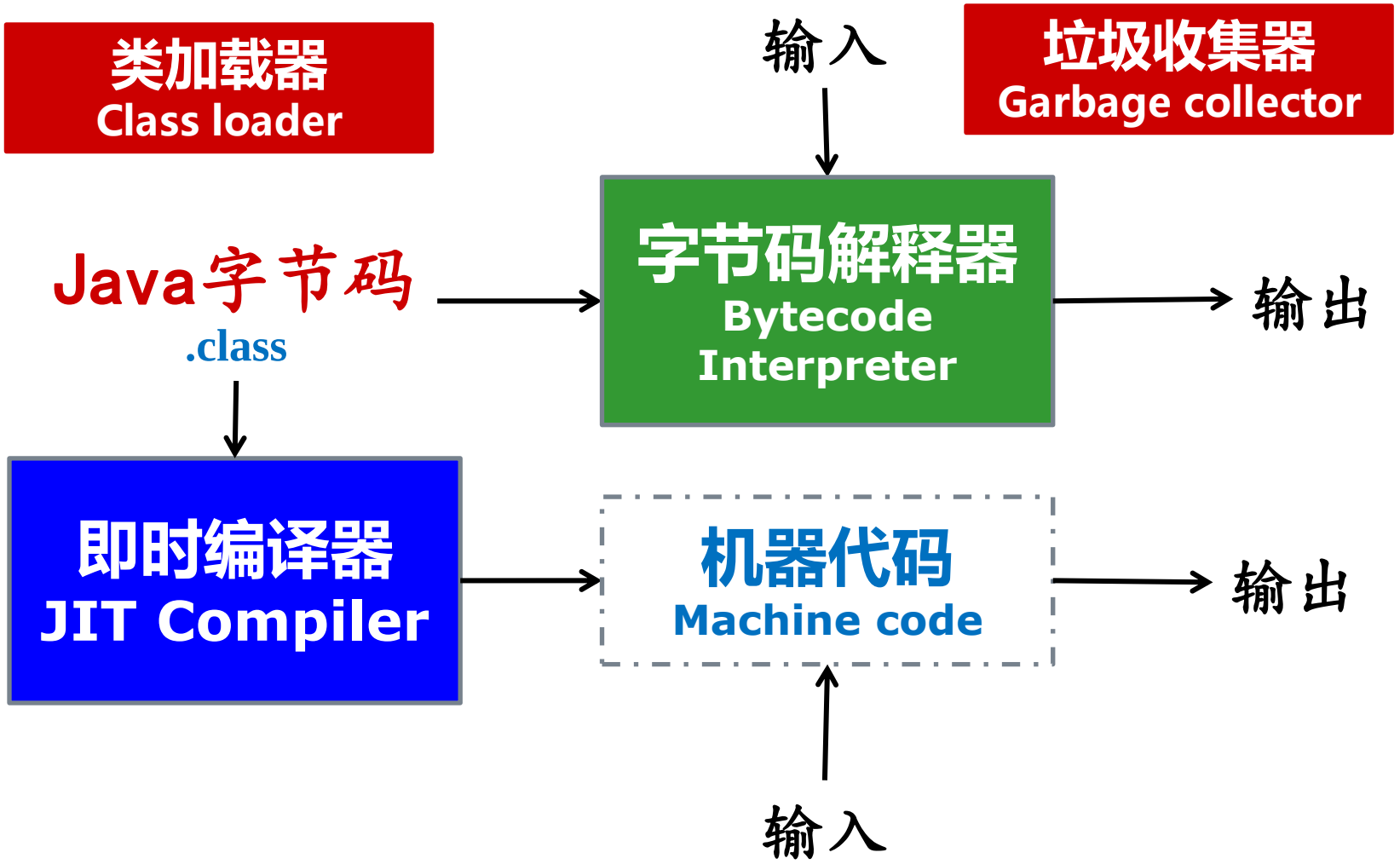

源程序
.java
翻译器
Translator
(javac)
中间表示
Intermediate Representation
.class
输入
Java虚拟机
Virtual Machine
(java)
输出

编译器的其他形式

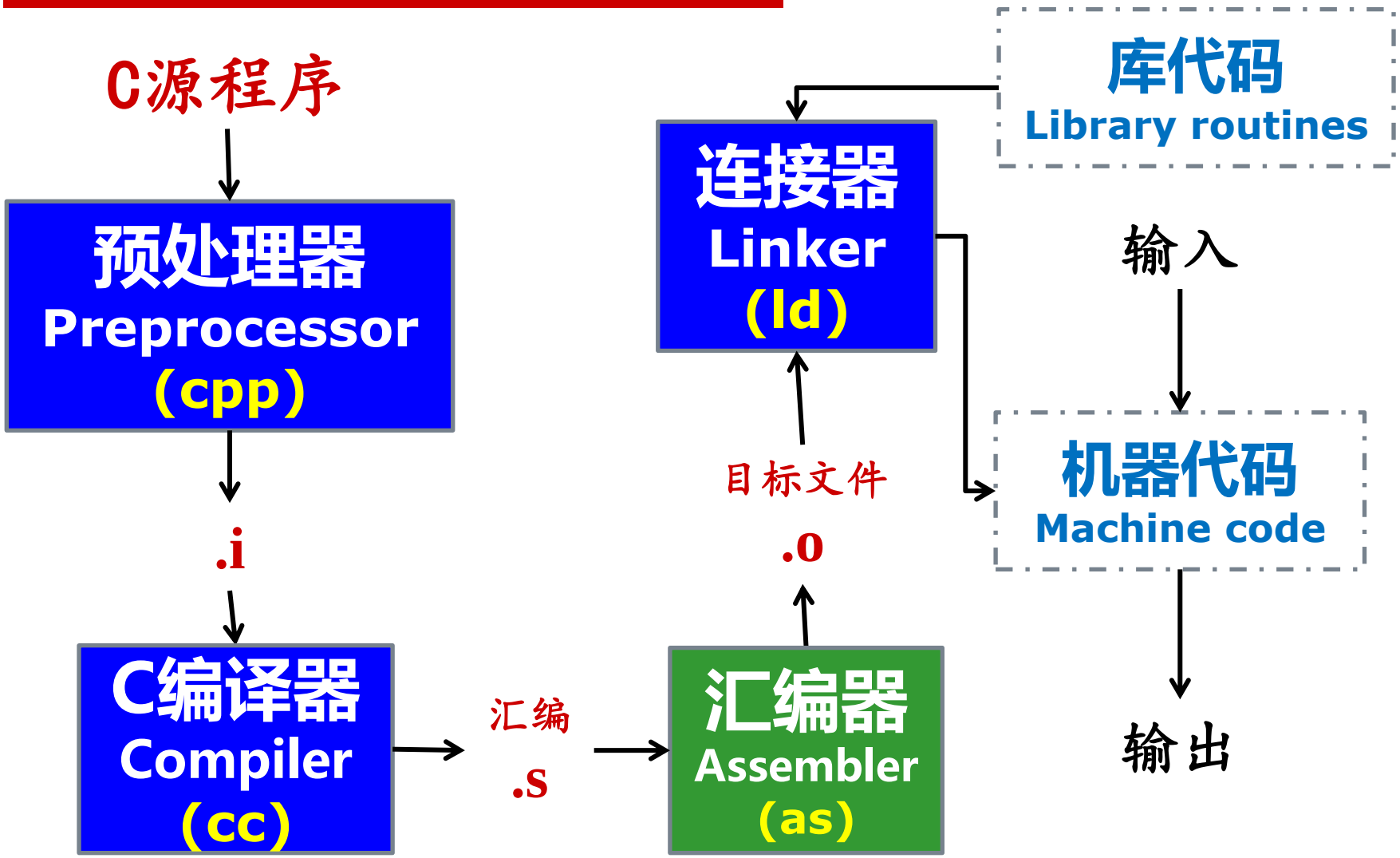
- 交叉编译器（Cross compiler）
  - 在一个平台上生成另一个平台上的代码

    PC → arm-linux-gcc → ARM
- 增量编译器（Incremental compiler）
  - 以增量地编译源程序,只编译修改的部分, 如 Freeline
- 即时编译器（Just-in-time compiler）
  - 在运行时对IR中每个被调用的方法进行编译，得到目标机器的本地代码，如 Java VM 中的即时编译器
- 预先编译器（Ahead-of-time compiler ）
  - 在程序执行之前将IR翻译成本地码，如 ART中的AOT

Java虚拟机


类加载器
Class loader
输入
垃圾收集器
Garbage collector
Java字节码
.class
字节码解释器
Bytecode
Interpreter
输出
即时编译器
JIT Compiler
机器代码
Machine code
输出
输入

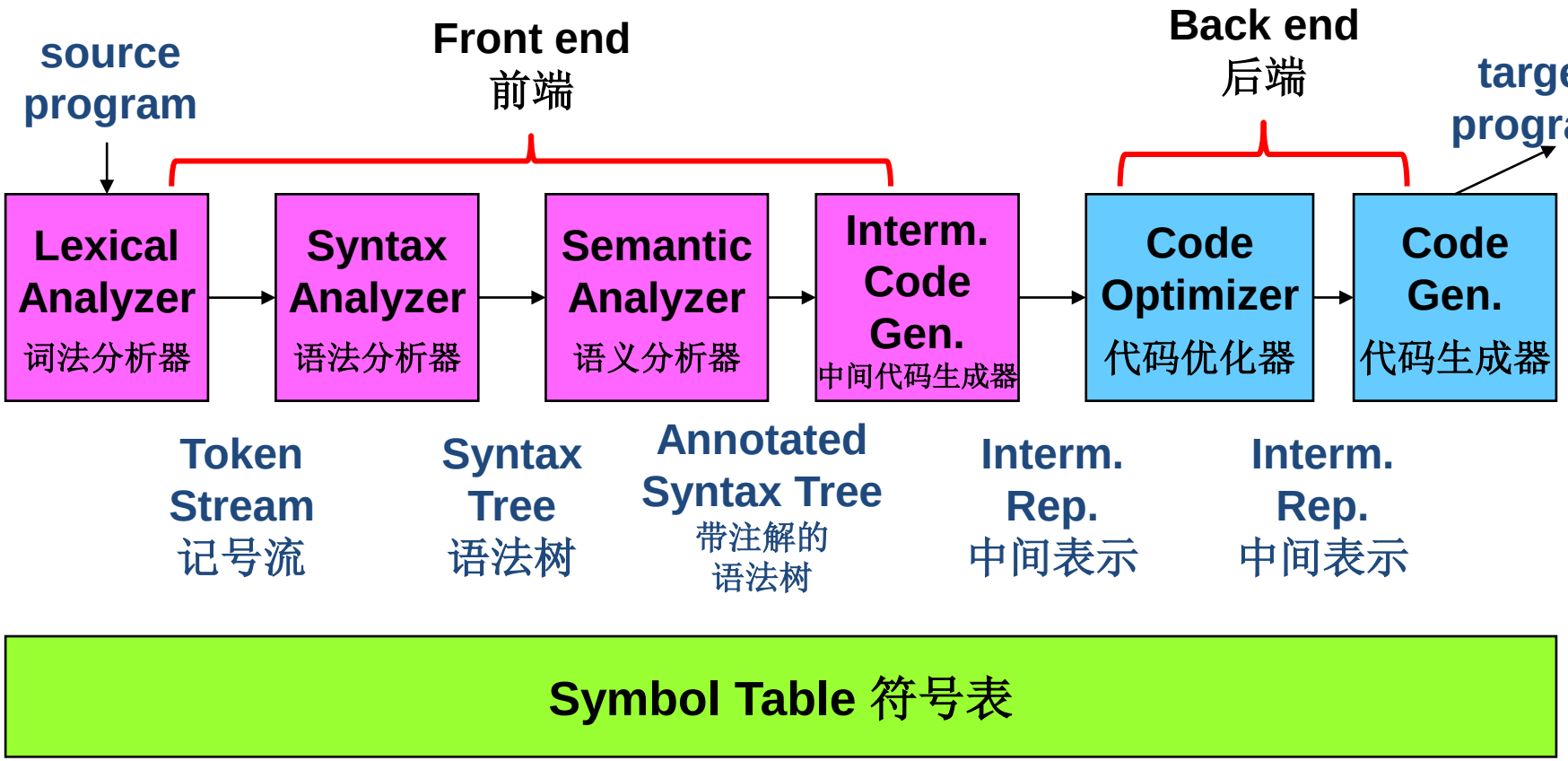
C 编译器


C源程序
预处理器
Preprocessor
(cpp)
.i
C编译器
Compiler
(cc)
汇编
.s
汇编器
Assembler
(as)
目标文件
.o
连接器
Linker
(ld)
库代码
Library routines
输入
机器代码
Machine code
输出

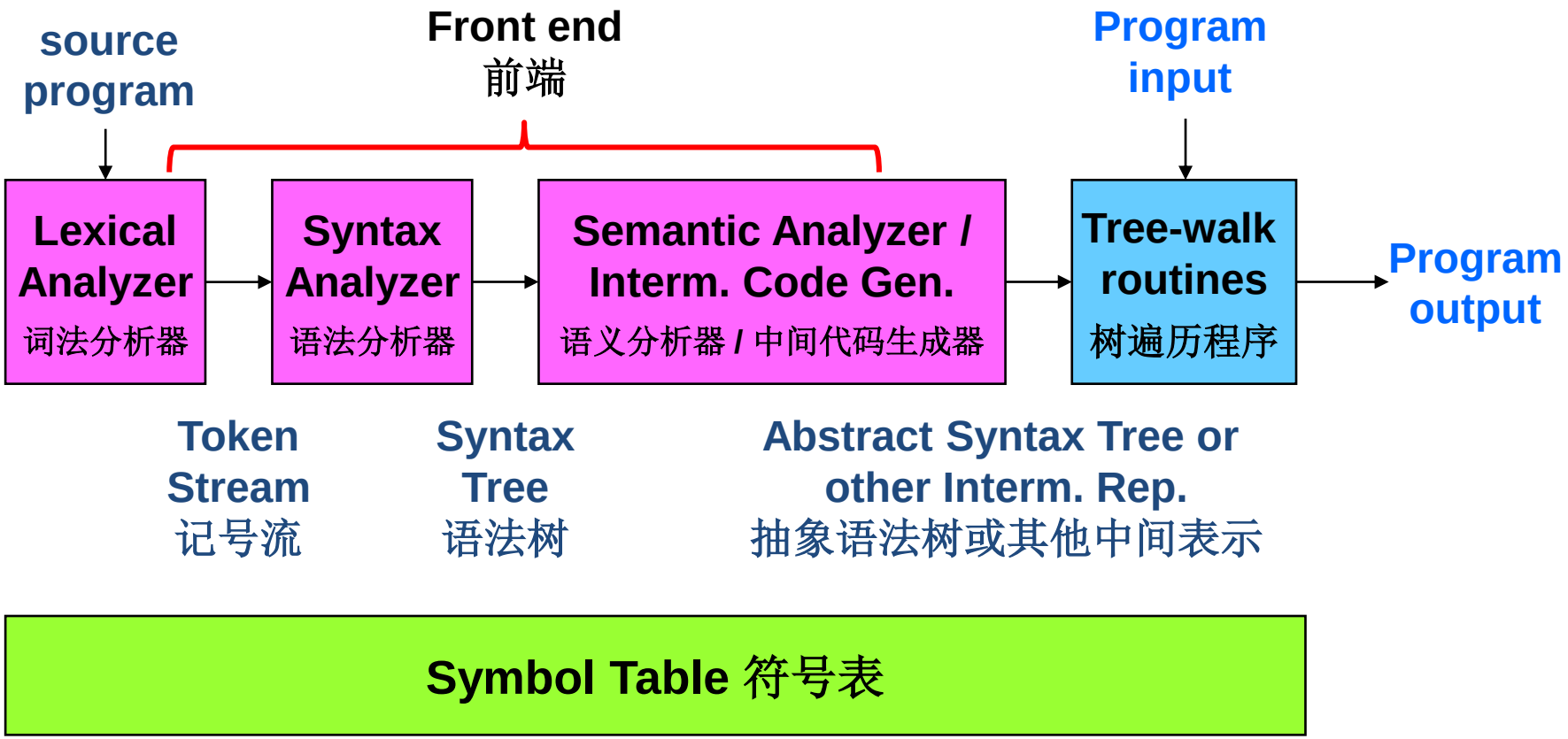
主要内容

1. 编程语言及设计
2. 编译器及形式
3. **编译器的阶段**
4. 示例：程序的表示
5. 基础实验的考虑

编译器的阶段

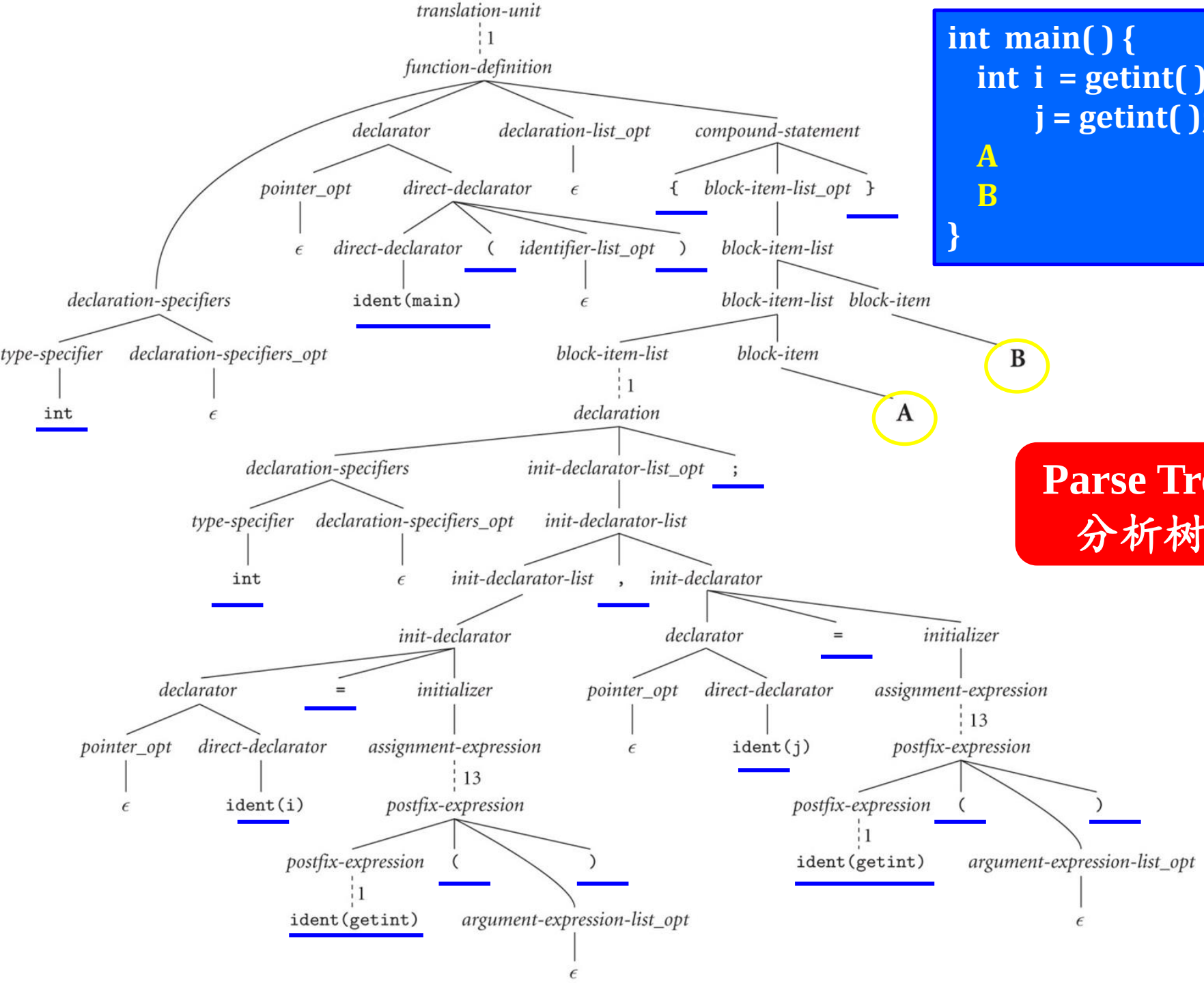

source program
Front end
前端
Back end
后端
target program
Lexical Analyzer
词法分析器
Syntax Analyzer
语法分析器
Semantic Analyzer
语义分析器
Interm. Code Gen.
中间代码生成器
Code Optimizer
代码优化器
Code Gen.
代码生成器
Token Stream
记号流
Syntax Tree
语法树
Annotated Syntax Tree
带注解的语法树
Interm. Rep.
中间表示
Interm. Rep.
中间表示
Symbol Table 符号表

编译器的阶段


source program
Front end
前端
Program input
Lexical Analyzer
词法分析器
Syntax Analyzer
语法分析器
Semantic Analyzer / Interm. Code Gen.
语义分析器 / 中间代码生成器
Tree-walk routines
树遍历程序
Program output
Token Stream
记号流
Syntax Tree
语法树
Abstract Syntax Tree or other Interm. Rep.
抽象语法树或其他中间表示
Symbol Table 符号表

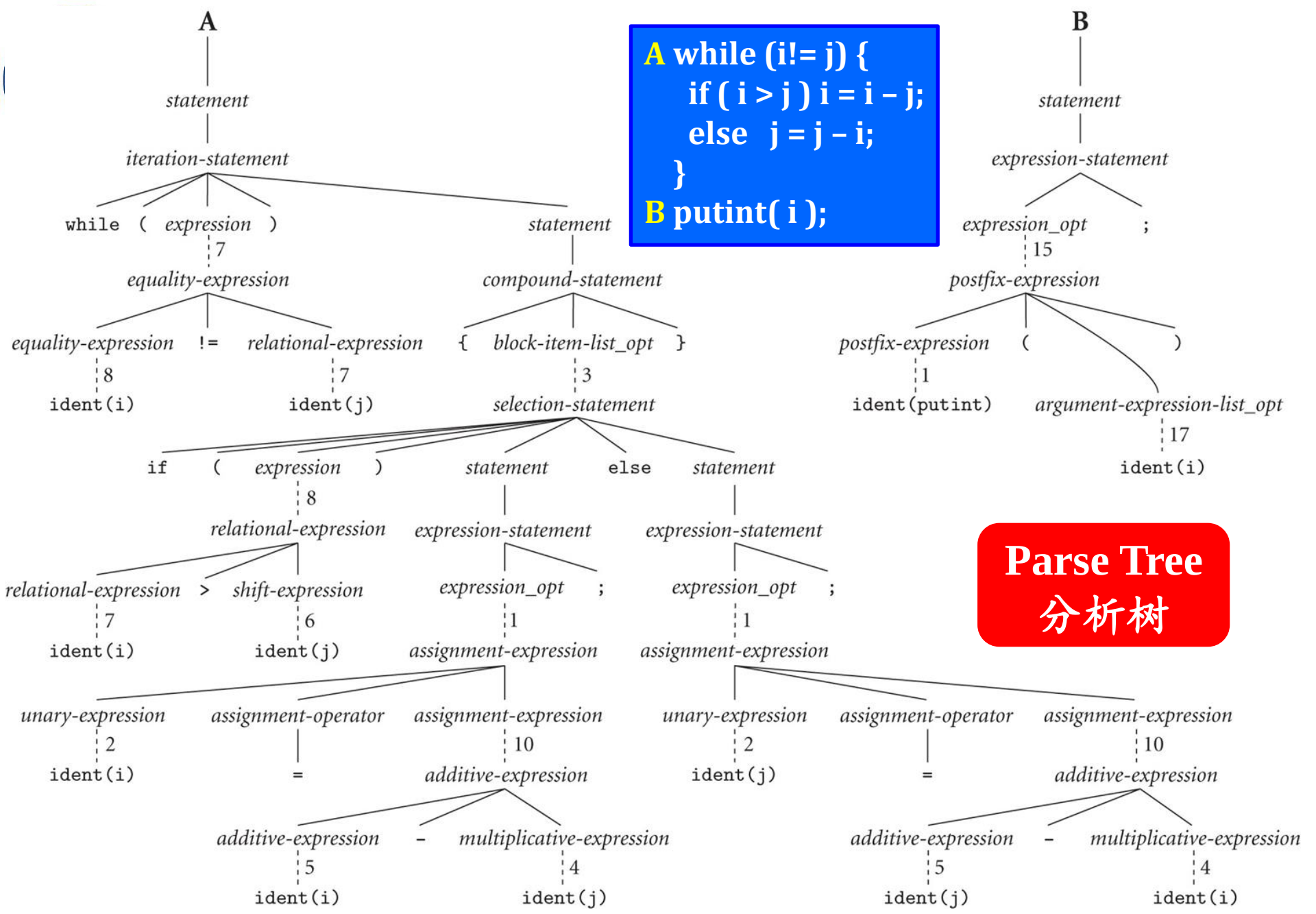
主要内容

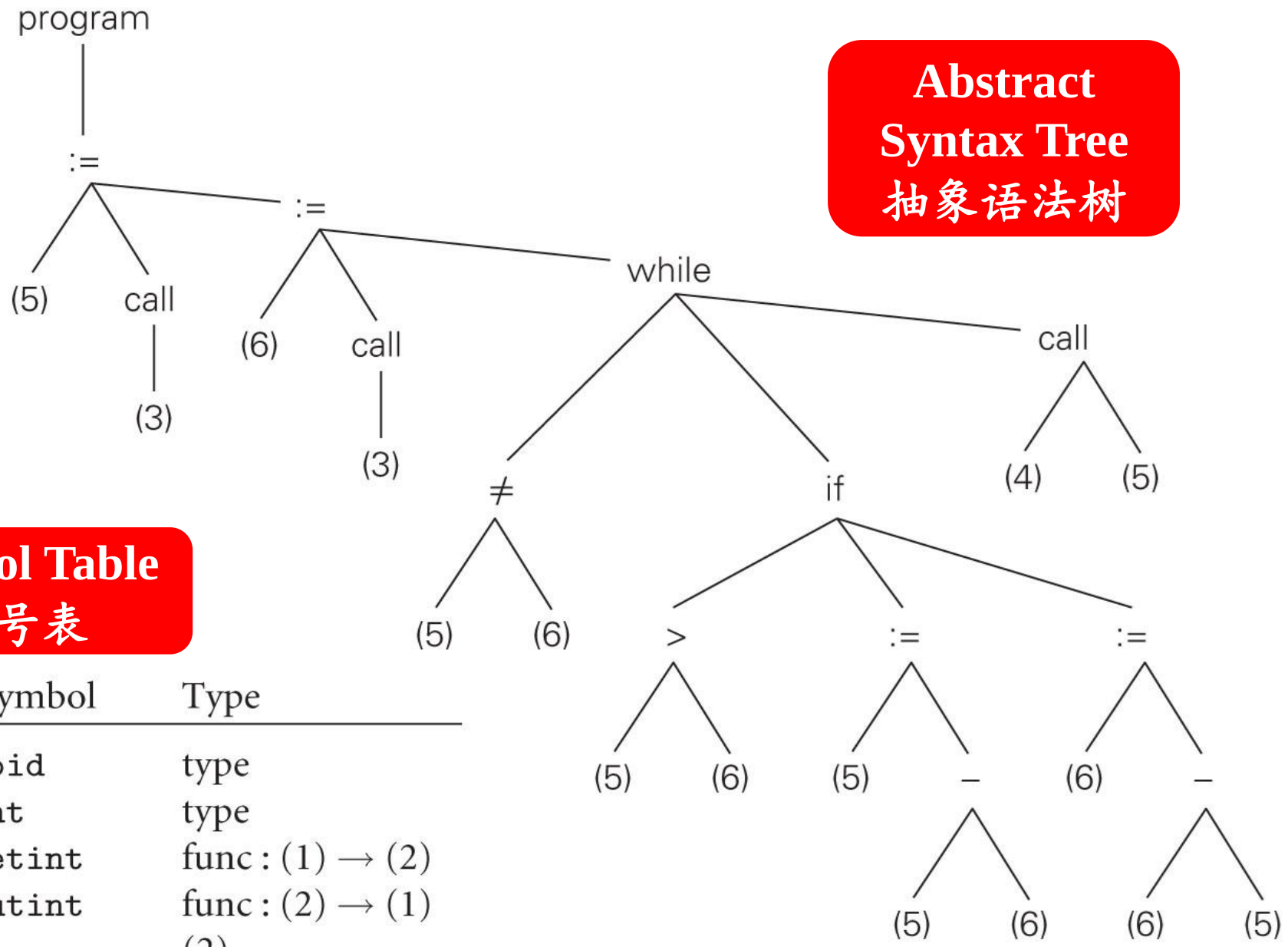
1. 编程语言及设计
2. 编译器及形式
3. 编译器的阶段
4. **示例：程序的表示**
5. 基础实验的考虑




```
int main( ) {
   int i = getint( ),
       j = getint( );
   A
   B
}
```

Parse Tree
分析树


A while (i!= j) {
if (i > j) i = i – j;
else j = j – i;
}
B putint(i);
A
B
statement
iteration-statement
while (expression) statement
7
equality-expression
equality-expression != relational-expression
compound-statement
{ block-item-list_opt }
8
7
3
ident(i)
ident(j)
selection-statement
if (expression) statement else statement
8
relational-expression
relational-expression > shift-expression
expression-statement
expression_opt ;
7
6
1
ident(i)
ident(j)
assignment-expression
unary-expression assignment-operator assignment-expression
2
10
ident(i)
=
additive-expression
additive-expression – multiplicative-expression
5
4
ident(i)
ident(j)
expression-statement
expression_opt ;
1
assignment-expression
unary-expression assignment-operator assignment-expression
2
10
ident(j)
=
additive-expression
additive-expression – multiplicative-expression
5
4
ident(j)
ident(i)
expression-statement
expression_opt ;
15
postfix-expression
postfix-expression ( )
1
ident(putint)
argument-expression-list_opt
17
ident(i)
Parse Tree
分析树



Abstract Syntax Tree
抽象语法树

Symbol Table
符号表

Index	Symbol	Type
1	`void`	type
2	`int`	type
3	`getint`	func : (1) → (2)
4	`putint`	func : (2) → (1)
5	`i`	(2)
6	`j`	(2)

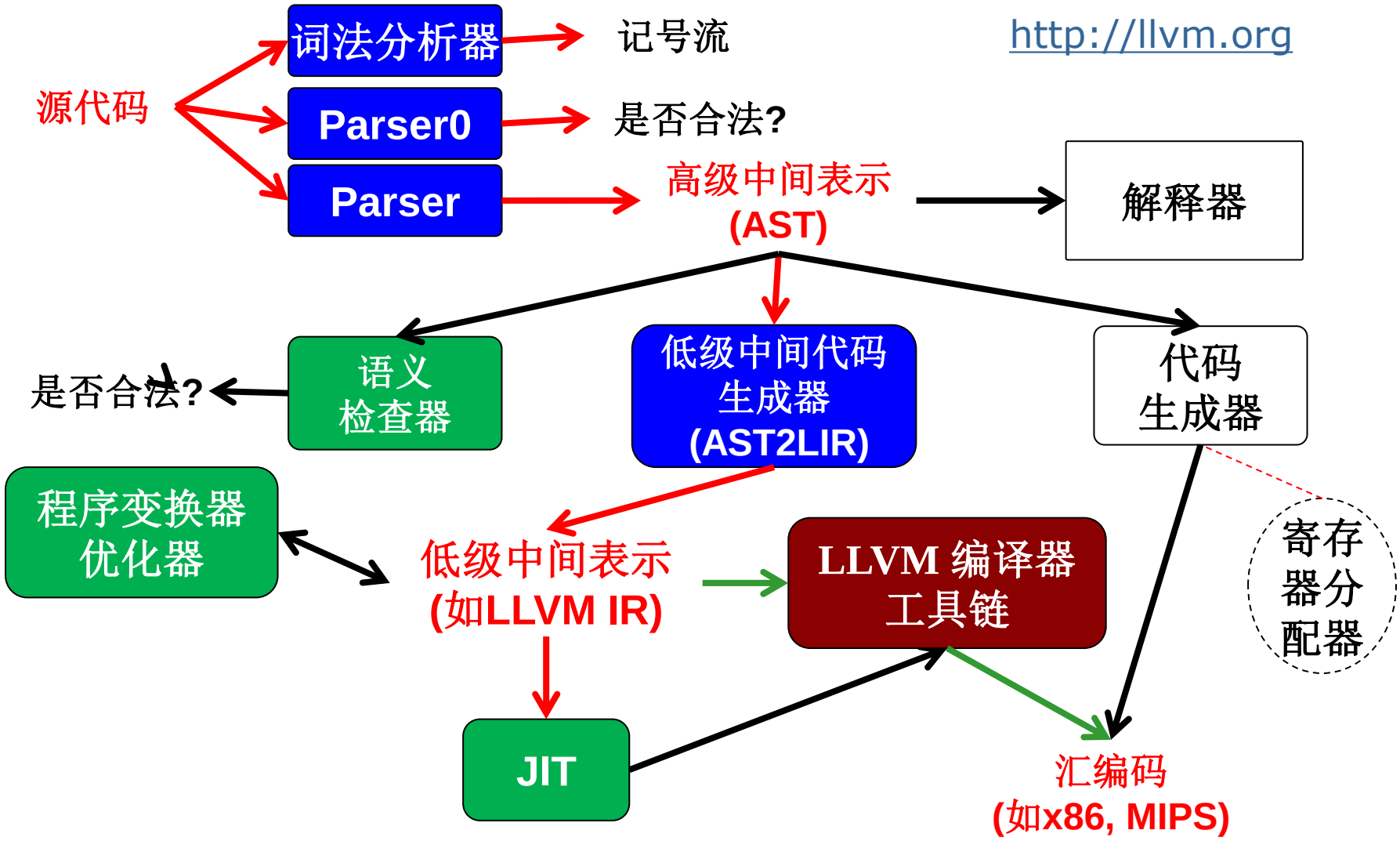
```
        pushl   %ebp                    # \
        movl    %esp, %ebp              #  ) reserve space for local variables
        subl    $16, %esp               # /
        call    getint                  # read
        movl    %eax, -8(%ebp)          # store i
        call    getint                  # read
        movl    %eax, -12(%ebp)         # store j
A:      movl    -8(%ebp), %edi          # load i
        movl    -12(%ebp), %ebx         # load j
        cmpl    %ebx, %edi              # compare
        je      D                       # jump if i == j
        movl    -8(%ebp), %edi          # load i
        movl    -12(%ebp), %ebx         # load j
        cmpl    %ebx, %edi              # compare
        jle     B                       # jump if i < j
        movl    -8(%ebp), %edi          # load i
        movl    -12(%ebp), %ebx         # load j
        subl    %ebx, %edi              # i = i - j
        movl    %edi, -8(%ebp)          # store i
        jmp     C
B:      movl    -12(%ebp), %edi         # load j
        movl    -8(%ebp), %ebx          # load i
        subl    %ebx, %edi              # j = j - i
        movl    %edi, -12(%ebp)         # store j
C:      jmp     A
D:      movl    -8(%ebp), %ebx          # load i
        push    %ebx                    # push i (pass to putint)
        call    putint                  # write
        addl    $4, %esp                # pop i
        leave                           # deallocate space for local variables
        mov     $0, %eax                # exit status for program
        ret                             # return to operating system
```

```
int main( ) {
  int i = getint( ),
      j = getint( );
A while (i!= j) {
    if ( i > j ) i = i – j;
    else B j = j – i;
C  }
D putint( i );
}
```

主要内容

1. 编程语言及设计
2. 编译器及形式
3. 编译器的阶段
4. 示例：程序的表示
5. **基础实验的考虑**

基础实验的考虑


词法分析器
记号流
http://llvm.org
源代码
Parser0
是否合法?
Parser
高级中间表示
(AST)
解释器
语义
检查器
是否合法?
低级中间代码
生成器
(AST2LIR)
代码
生成器
程序变换器
优化器
低级中间表示
(如LLVM IR)
LLVM 编译器
工具链
寄存
器分
配器
JIT
汇编码
(如x86, MIPS)

解析器的生成器

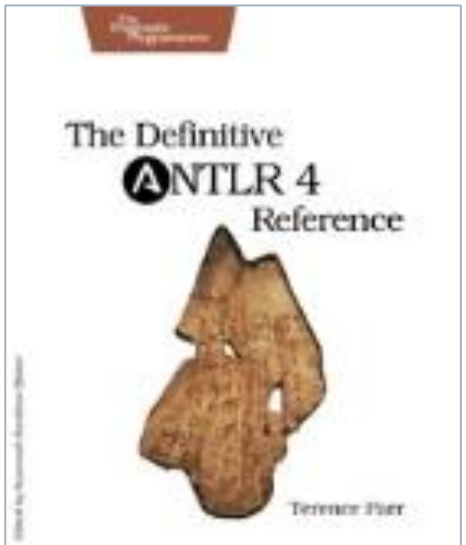
- 生成器
  - 生成Lexer：Flex（for windows）、Jflex
  - 生成Parser
    - LALR: Bison（for windows）、Java CUP
    - LL: JavaCC、ANTLR – LL(*) [PLDI2011]
- 文法对Parser的影响
  - LR Parser的优势：速度快、表达能力强
  - LL Parser的优势：代码结构与文法对应，易理解，容易增加错误处理和错误恢复

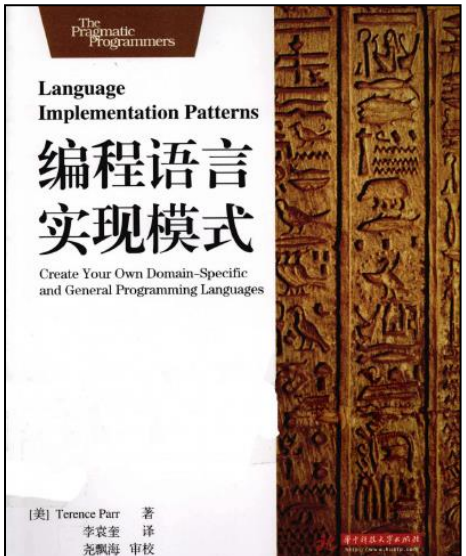
ANTLR(ANother Tool for Language Recognition)

[http://www.antlr.org/]

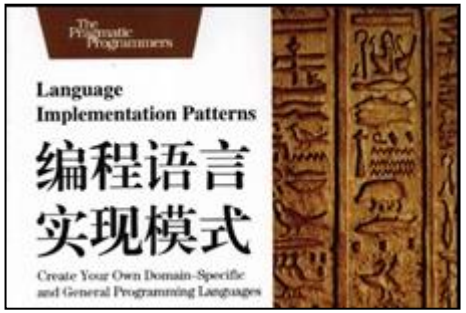
- Prof. Terence Parr，since 1989
- 支持多种代码生成目标

  Java、C++、C#、Python、Go、JavaScript、Swift





基于ANTLR开展的实验



- 开源：源码阅读，消化、理解生成器基于的原理
- 各种语言实现：31种模式

1. 从文法到递归下降识别器
2. LL(1)递归下降词法分析器
3. LL(1)递归下降语法解析器
4. LL(k)递归下降语法解析器
5. 回溯解析器
6. 记忆解析器
7. 谓词解析器
8. 解析树（分析树）
9. 同型抽象语法树
10. 规范化异型AST
11. 不规则的异型AST
12. 内嵌遍历器
13. 外部访问者
14. 文法访问者
15. 模式匹配者（子树）
16. 单作用域符号表
17. 嵌套作用域符号表
18. 数据聚集的符号表
19. 类的符号表型
20. 计算表达式的类型
21. 自动类型提升
22. 静态类型检查
23. 动态类型检查
24. 语法制导的解释器
25. 基于树的解释器
26. 字节码汇编器
27. 栈式解释器
28. 寄存器解释器
29. 语法制导的翻译器
30. 基于规则的翻译器
31. 模型驱动的转换

我听到的会忘掉，

我看到的能记住，

我做过的才真正明白。