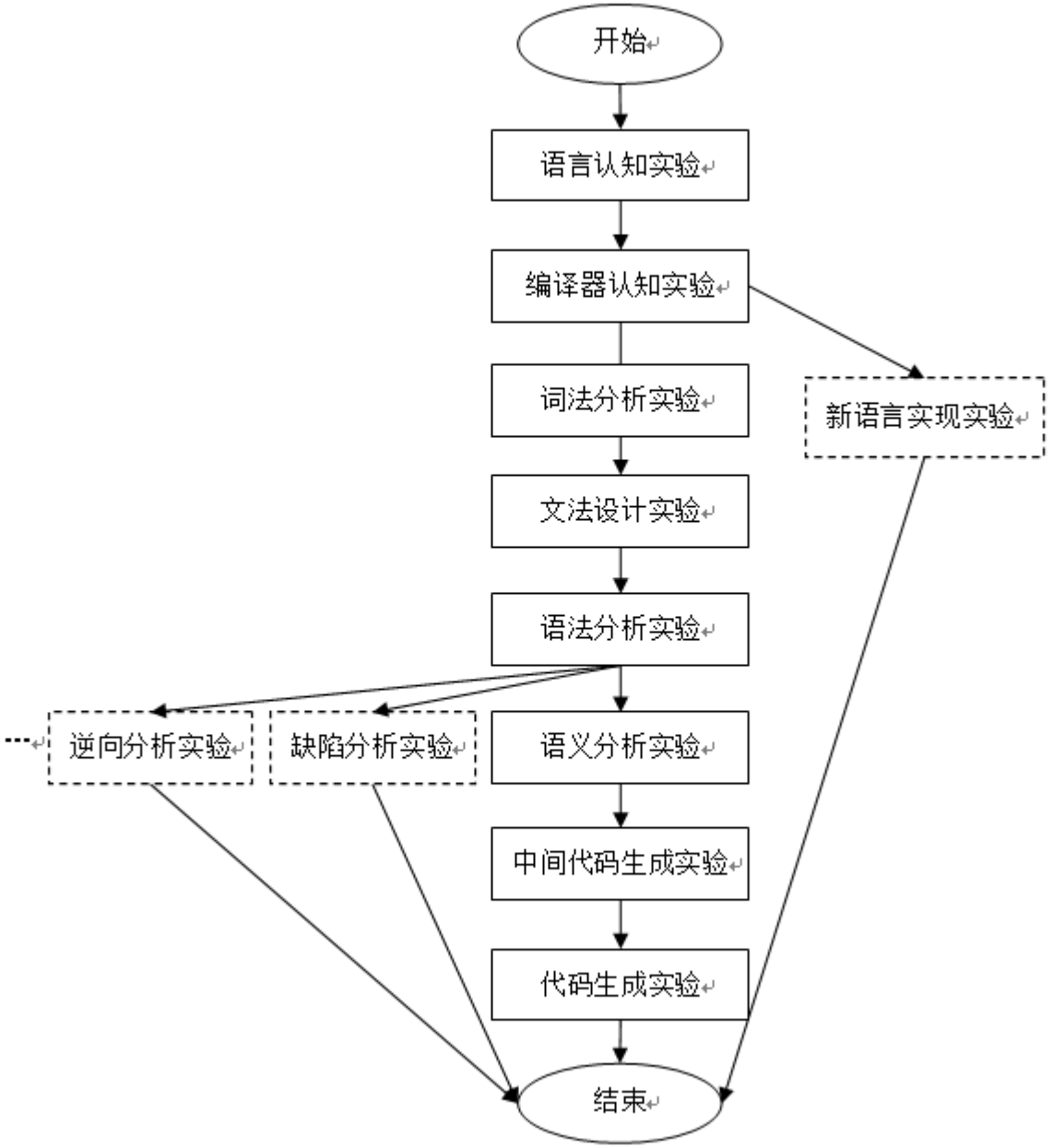
BIT-MiniCC简介

北京理工大学计算机学院

总体实验框架设计


开始
语言认知实验
编译器认知实验
词法分析实验
文法设计实验
语法分析实验
逆向分析实验
缺陷分析实验
语义分析实验
中间代码生成实验
代码生成实验
结束
新语言实现实验

提纲

1. BIT-MiniCC简介
2. 框架介绍
3. 框架使用方法
4. Q&A

BIT-MiniCC简介

- 课程实验存在的问题
  - 从理论到实践的距离：我听明白了，但是还是不知道怎么实现
  - 从前端到后端的距离：词法分析实现了，但是不怎么好
  - 从理想到现实的距离：老师，我这学期有5门课。。。

大作业A？

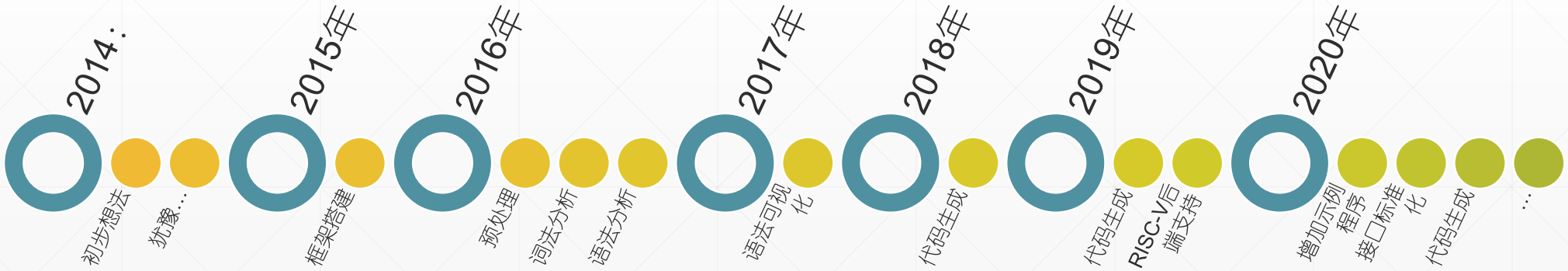
大作业B？

大作业C？

竞赛、实验室项目？

BIT-MiniCC简介

- BIT-MiniCC = **BIT Mini C C**ompiler = 一个迷你C语言编译器
  - 源语言　：C语言子集
  - 目标语言：MIPS汇编语言，RISC-V汇编语言，X86汇编语言
  - 宿主语言：Java / Java+C / Java+C# / Java+Python / Java+…
- 开发现状

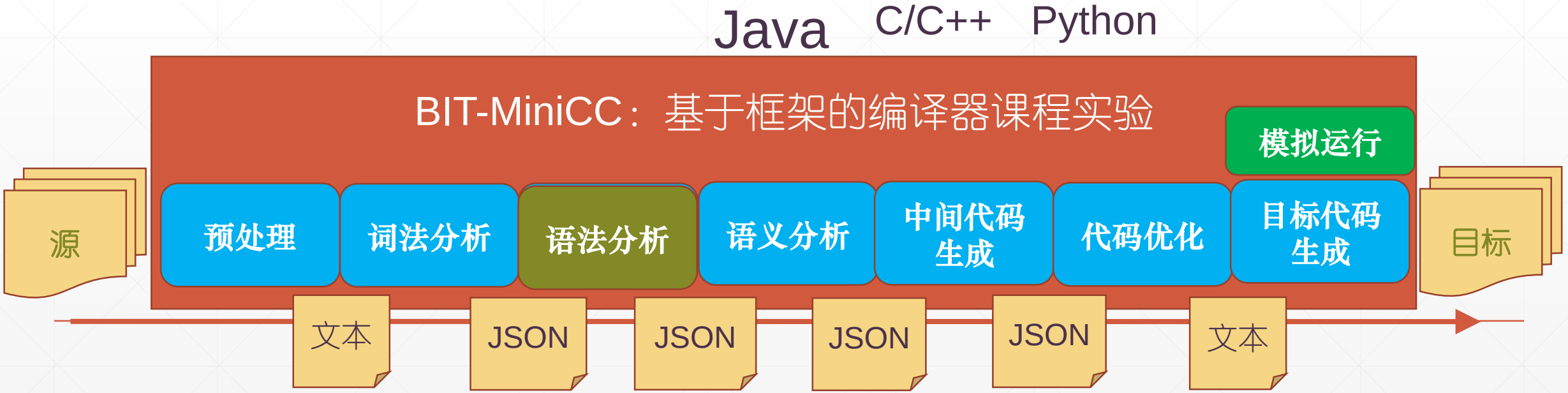


BIT-MiniCC简介

- 设计目标
  - 提供一个可参考的运行实例
    - 采用熟悉的语言实现：Java、C#和Python等
    - 内部集成了各个部分的实现
    - 中间处理结果可见：JSON等
    - 每个过程可替换：前面做的不好没关系，可以用现有的实现
  - 除此之外
    - 框架源码公开，但是内部实现不对外公开
    - 代码复制检测： 框架能极大降低检测范围
    - 最后结果测试自动化： 标准输入输出使得自动测试成为可能

BIT-MiniCC：基于框架的实验设计

- 设计目标1：跨平台可运行参考实例
- 设计目标2：框架源码可见，中间结果可见
- 设计目标3：核心模块不可见，可替换
- 设计目标4：支持多种语言插接
- 设计目标5：充分基于开源库设计并实现

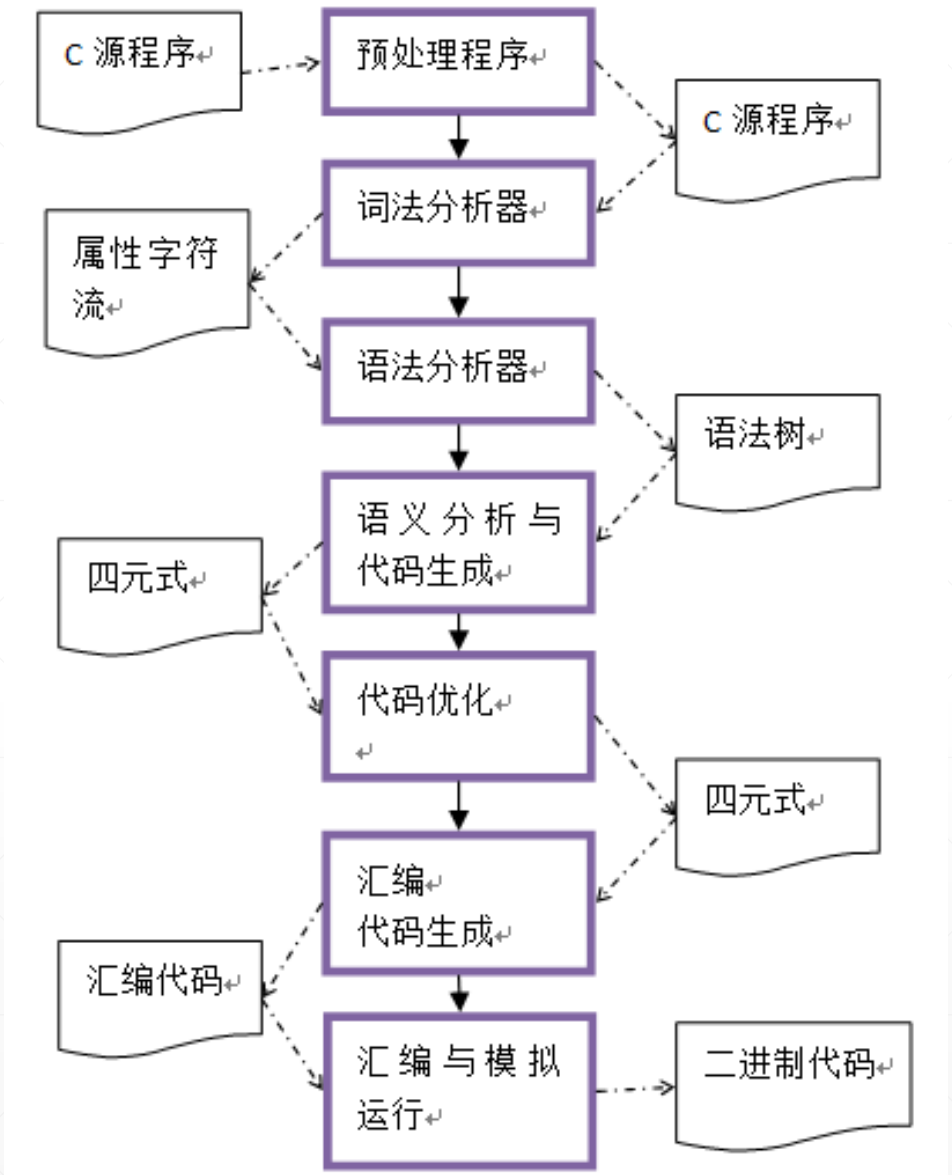


BIT-MiniCC：基于框架的实验设计

- 设计目标1：跨平台可运行参考实例
- 设计目标2：框架源码可见，中间结果可见
- 设计目标3：核心模块不可见，可替换
- 设计目标4：支持多种语言插接

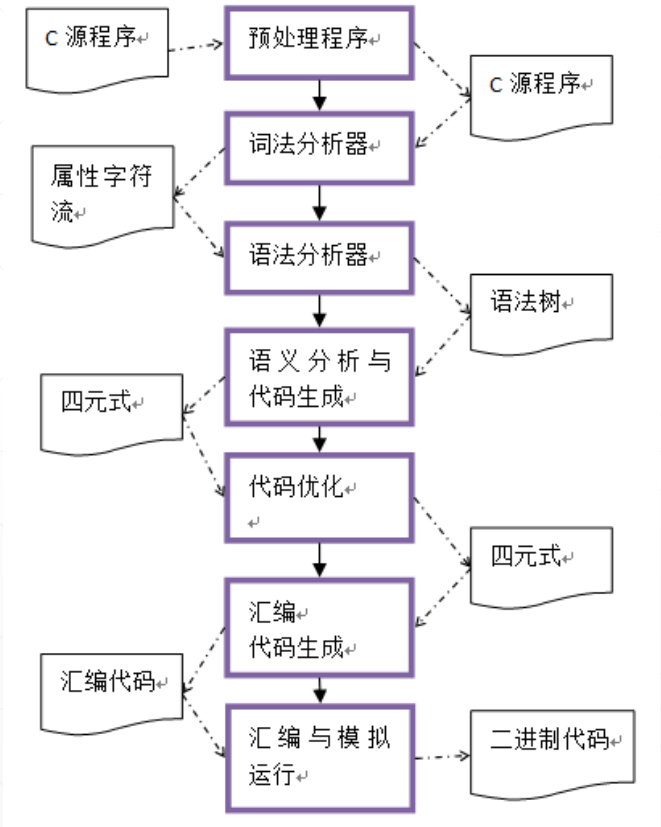
框架介绍

- 整体结构


C 源程序
预处理程序
C 源程序
词法分析器
属性字符流
语法分析器
语法树
语义分析与代码生成
四元式
代码优化
四元式
汇编代码生成
汇编代码
汇编与模拟运行
二进制代码

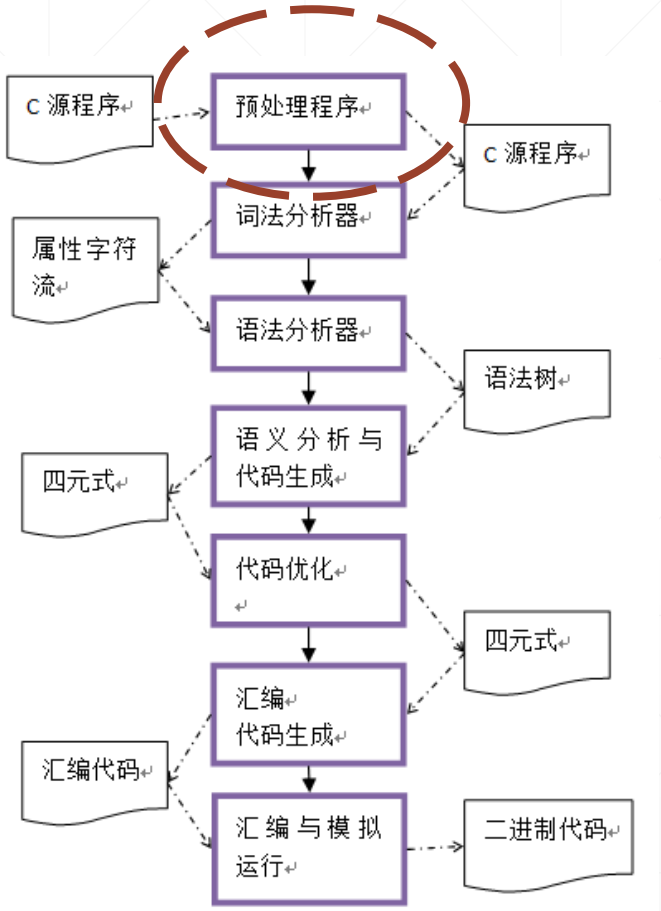
框架介绍

- 特点
  - 输入输出标准化
  - 单个模块可替换
  - 只需关注单个模块的设计
  - 内部模块集成实现
  - 内部实现和自主实现可组合



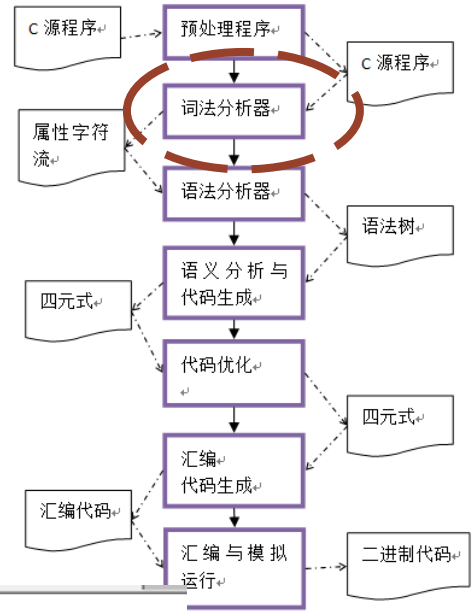
框架介绍

- 预处理
  - 输入：源程序
  - 输出：处理后的源程序
  - 功能
    - 文件包含
    - 宏替换
    - 删除注释
    - 无用空白删除



框架介绍

- 词法分析
  - 输入：清理后的源程序
  - 输出：属性字符流
  - 功能：根据词法规则识别C语言的全部单词类别，输出相应的属性字符流



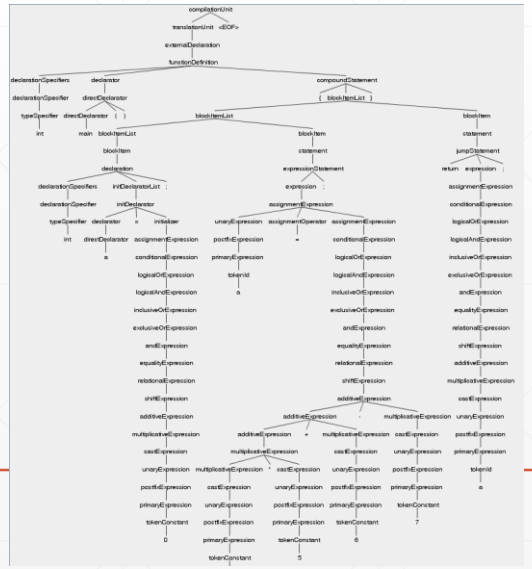

```
int sum(int a, int b) { return a + b; }
```

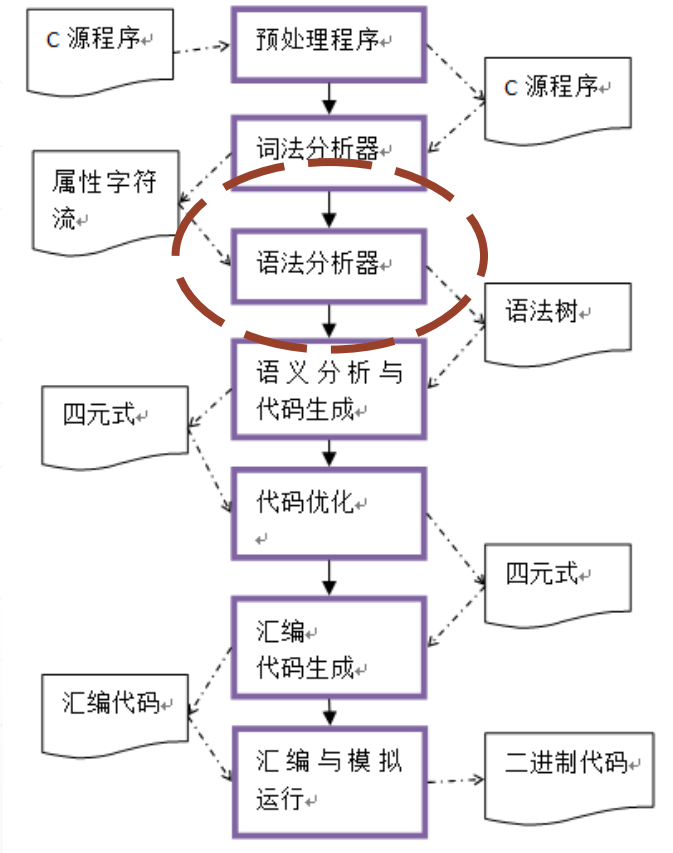
```
[@0,0:2='int',<'int'>,1:0]
[@1,4:6='sum',<Identifier>,1:4]
[@2,8:8='(',<'('>,1:8]
[@3,10:12='int',<'int'>,1:10]
[@4,14:14='a',<Identifier>,1:14]
[@5,16:16=',',<','>,1:16]
[@6,18:20='int',<'int'>,1:18]
[@7,22:22='b',<Identifier>,1:22]
[@8,24:24=')',<')'>,1:24]
[@9,26:26='{',<'{'>,1:26]
[@10,28:33='return',<'return'>,1:28]
[@11,35:35='a',<Identifier>,1:35]
[@12,37:37='+',<'+'>,1:37]
[@13,39:39='b',<Identifier>,1:39]
[@14,41:41=';',<';'>,1:41]
[@15,43:43='}',<'}'>,1:43]
[@16,0:4='<EOF>',<EOF>,2:0]
```

框架介绍

- 语法分析
  - 输入：属性字符流
  - 输出：语法树
  - 功能：根据C语言的语法规则，识别输出程序的结构，
  - 输出语法错误或者语法树

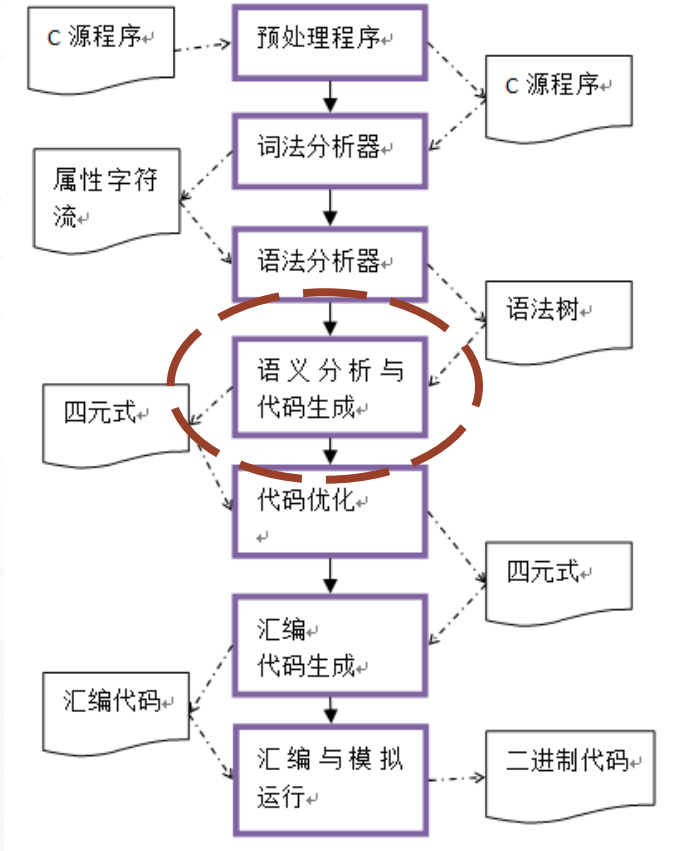
```
[@0,0:2='int',<'int'>,1:0]
[@1,4:6='sum',<Identifier>,1:4]
[@2,8:8='(',<'('>,1:8]
[@3,10:12='int',<'int'>,1:10]
[@4,14:14='a',<Identifier>,1:14]
[@5,16:16=',',<','>,1:16]
[@6,18:20='int',<'int'>,1:18]
[@7,22:22='b',<Identifier>,1:22]
[@8,24:24=')',<')'>,1:24]
[@9,26:26='{',<'{'>,1:26]
[@10,28:33='return',<'return'>,1:28]
[@11,35:35='a',<Identifier>,1:35]
[@12,37:37='+',<'+'>,1:37]
[@13,39:39='b',<Identifier>,1:39]
[@14,41:41=';',<';'>,1:41]
[@15,43:43='}',<'}'>,1:43]
[@16,0:4='<EOF>',<EOF>,2:0]
```





框架介绍

- 语义分析
  - 输入：语法树
  - 输出：语法树及语义错误
  - 功能
    - 检查符号是否重定义
    - 检查符号的作用域及引用
    - …

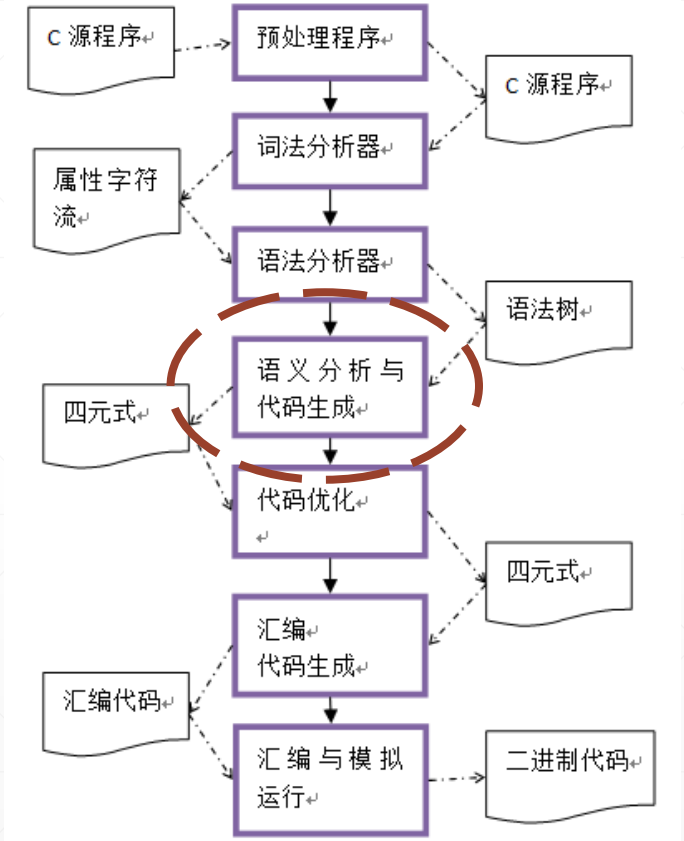


框架介绍

- 中间代码

and the statement "a = b + c - d" is:

```
dassign $a (
  sub i32(
   add i32(dread i32 $b, dread i32 $c),
   dread i32 $d))
```



框架介绍

- 代码生成
  - 生成目标机汇编代码

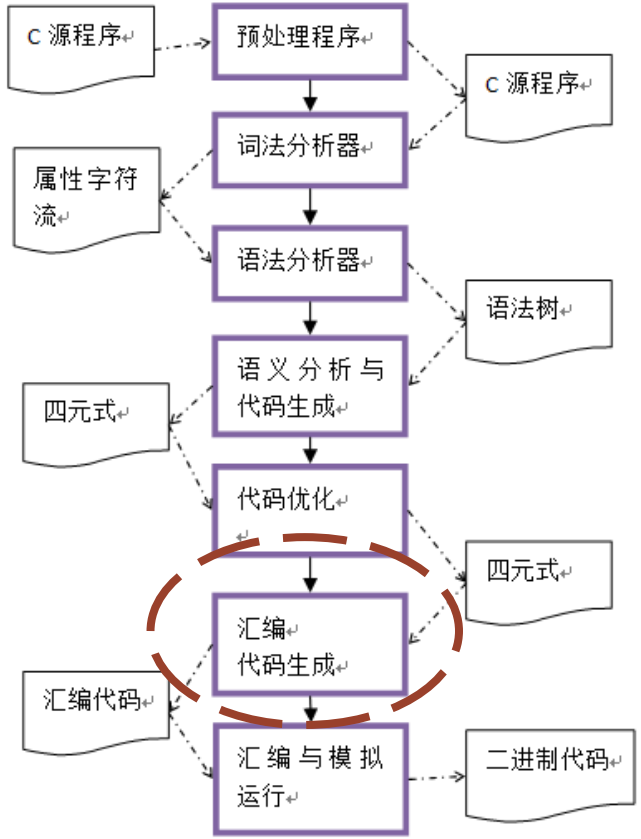
and the statement "a = b + c - d" is:

```
dassign $a (
  sub i32(
   add i32(dread i32 $b, dread i32 $c),
   dread i32 $d))
```

Edit Execute

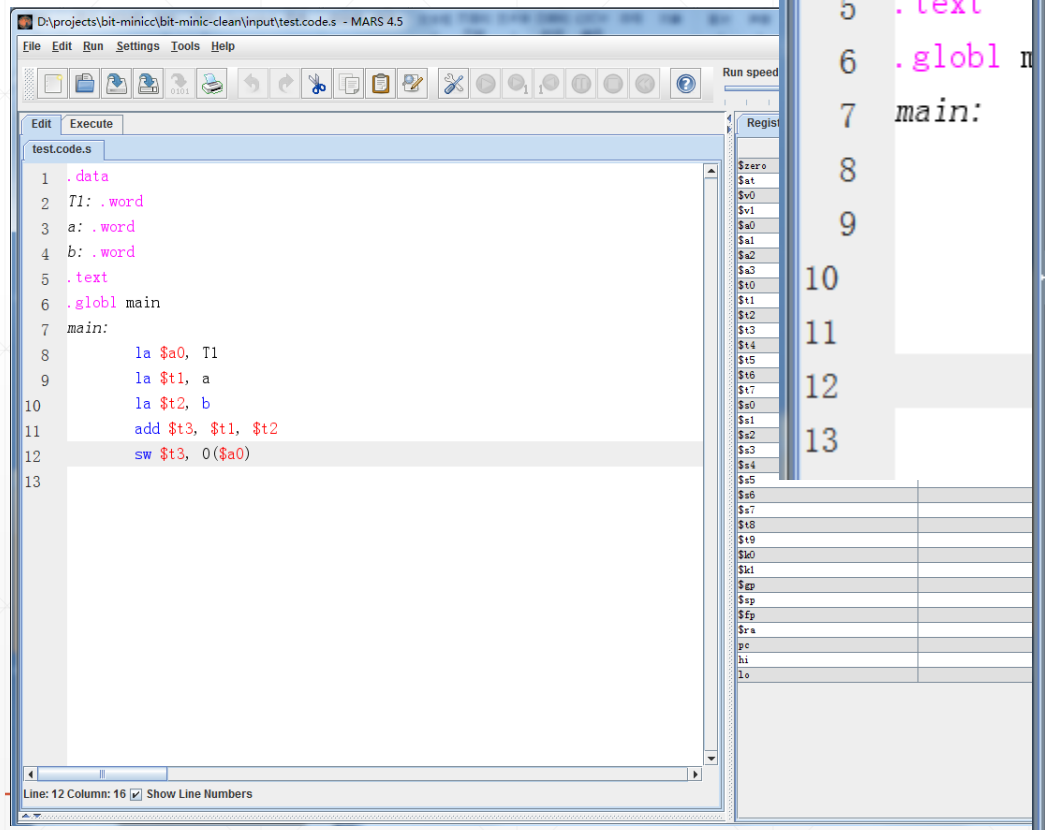
test.code.s

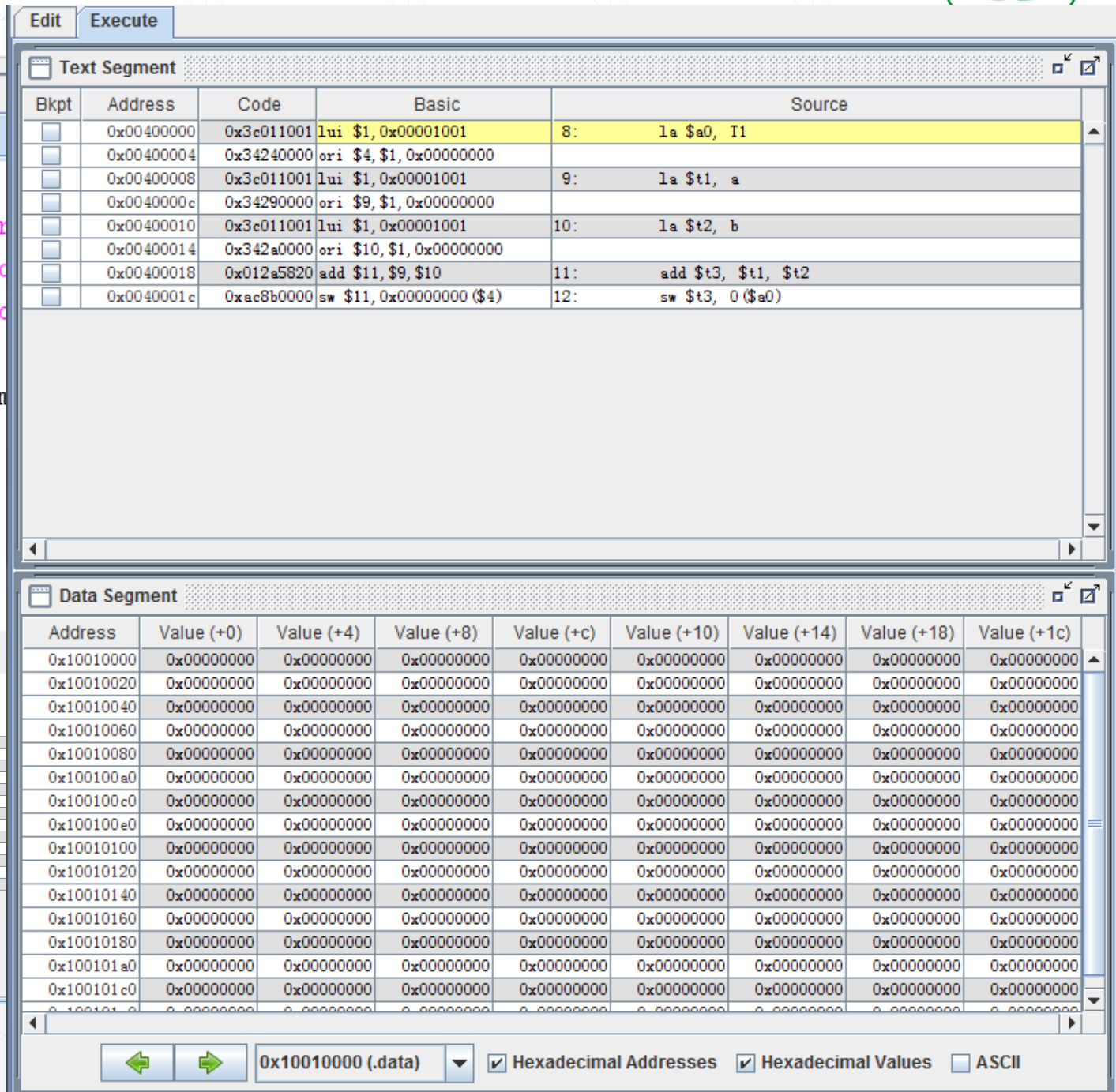
```
1  .data
2  T1: .word
3  a: .word
4  b: .word
5  .text
6  .globl main
7  main:
8          la $a0, T1
9          la $t1, a
10         la $t2, b
11         add $t3, $t1, $t2
12         sw $t3, 0($a0)
13
```



框架介绍

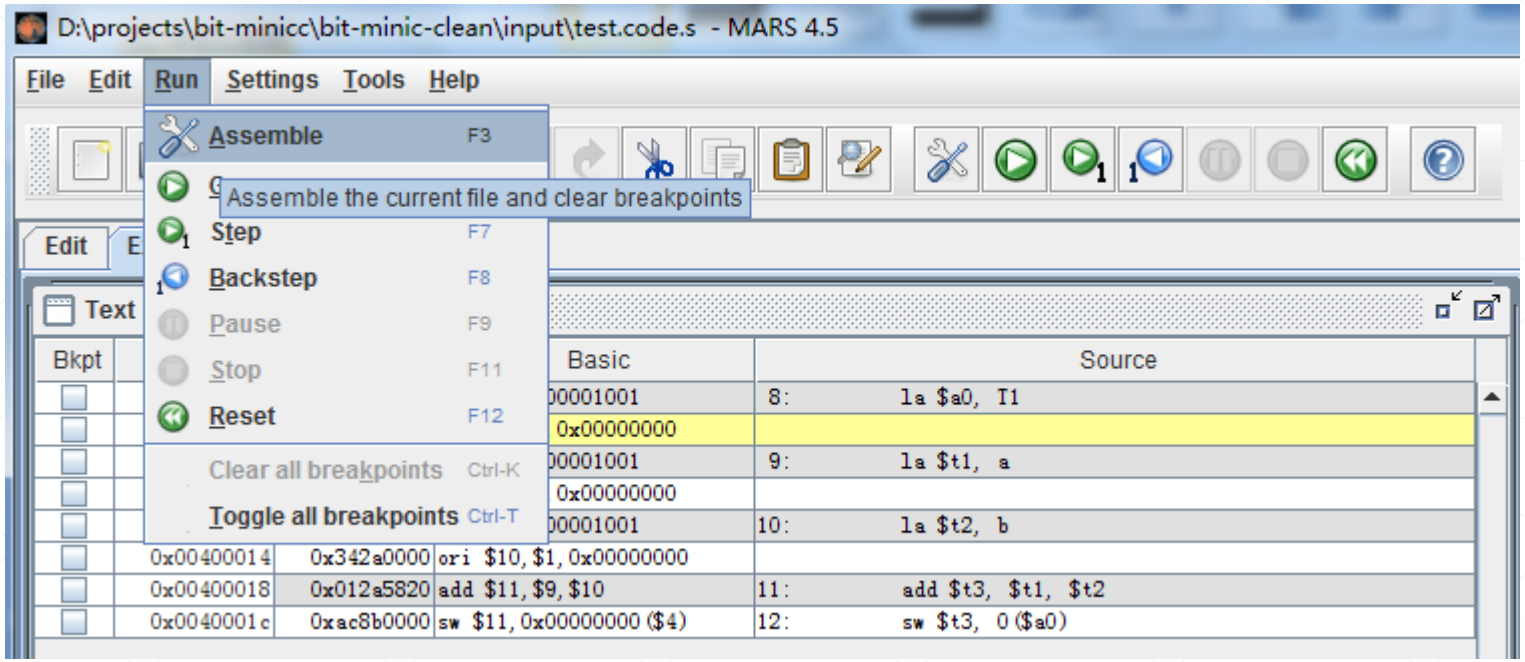
- 模拟运行： MARS

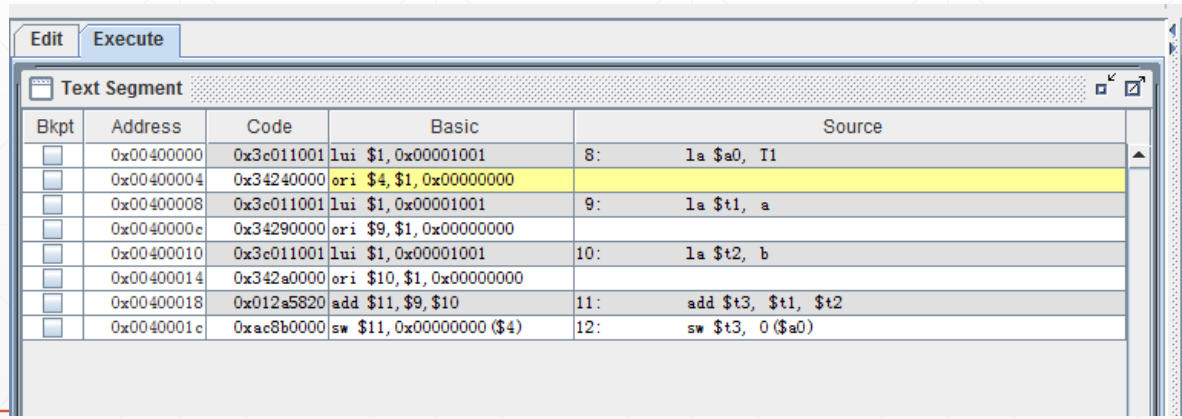


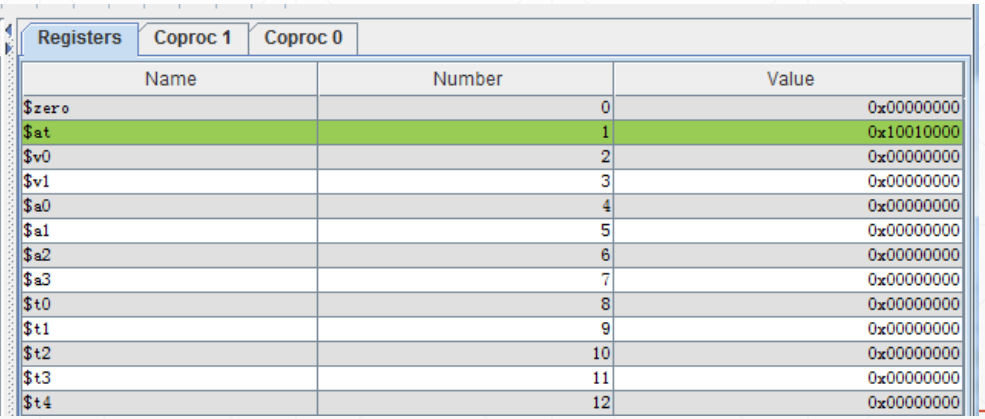


框架介绍

- 模拟运行

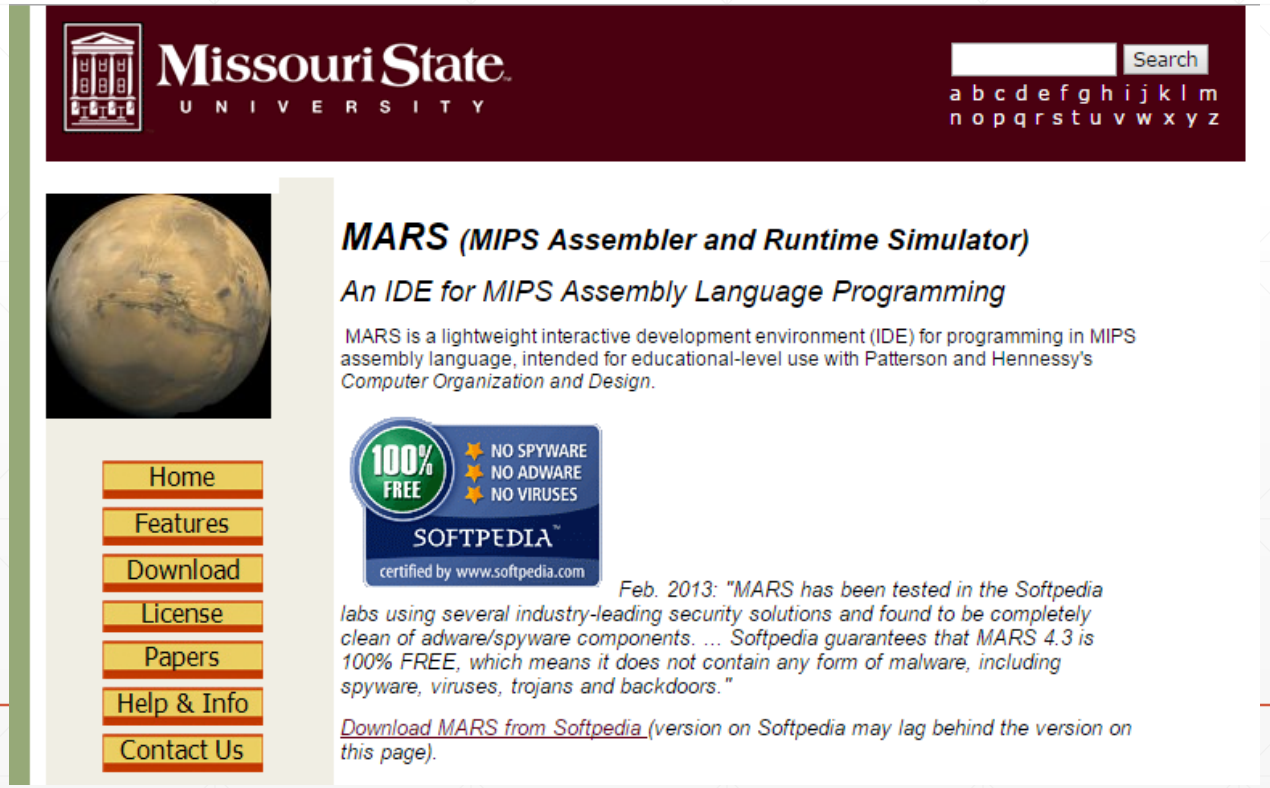

D:\projects\bit-minicc\bit-minic-clean\input\test.code.s - MARS 4.5
File Edit Run Settings Tools Help
Assemble F3
Assemble the current file and clear breakpoints
Step F7
Backstep F8
Pause F9
Stop F11
Reset F12
Clear all breakpoints Ctrl-K
Toggle all breakpoints Ctrl-T
Edit
Text
Bkpt
Basic
Source
0x00400014 0x342a0000 ori $10, $1, 0x00000000
0x00400018 0x012a5820 add $11, $9, $10 11: add $t3, $t1, $t2
0x0040001c 0xac8b0000 sw $11, 0x00000000 ($4) 12: sw $t3, 0 ($a0)


Edit Execute
Text Segment
Bkpt Address Code Basic Source
0x00400000 0x3c011001 lui $1, 0x00001001 8: la $a0, T1
0x00400004 0x34240000 ori $4, $1, 0x00000000
0x00400008 0x3c011001 lui $1, 0x00001001 9: la $t1, a
0x0040000c 0x34290000 ori $9, $1, 0x00000000
0x00400010 0x3c011001 lui $1, 0x00001001 10: la $t2, b
0x00400014 0x342a0000 ori $10, $1, 0x00000000
0x00400018 0x012a5820 add $11, $9, $10 11: add $t3, $t1, $t2
0x0040001c 0xac8b0000 sw $11, 0x00000000 ($4) 12: sw $t3, 0 ($a0)


Registers Coproc 1 Coproc 0
Name Number Value
$zero 0 0x00000000
$at 1 0x10010000
$v0 2 0x00000000
$v1 3 0x00000000
$a0 4 0x00000000
$a1 5 0x00000000
$a2 6 0x00000000
$a3 7 0x00000000
$t0 8 0x00000000
$t1 9 0x00000000
$t2 10 0x00000000
$t3 11 0x00000000
$t4 12 0x00000000

框架介绍

- MARS相关资料
  - http://courses.missouristate.edu/KenVollmar/MARS/



框架介绍

- 运行环境
  - java JRE ≥ 1.8
  - 各操作系统都可以

框架介绍

- 基本配置: config.xml
  - skip：是否跳过该阶段运行
  - type：模块实现方法
    - java/binary/python
  - path：路径
    - Binary
    - simulator
  - name：阶段名称

```
<?xml version="1.0" encoding="UTF-8"?>
<config name="config.xml">
  <phases>
    <phase>
      <phase skip="false" type="java" path="" name="pp" />
      <phase skip="false" type="java" path="" name="scanning" />
      <phase skip="false" type="java" path="" name="parsing" />
      <phase skip="false" type="java" path="" name="semantic" />
      <phase skip="false" type="java" path="" name="icgen" />
      <phase skip="false" type="java" path="" name="optimizing" />
      <phase skip="false" type="java" path="" name="codegen" />
      <phase skip="false" type="mips" path="" name="simulating" />
    </phase>
  </phases>
</config>
```

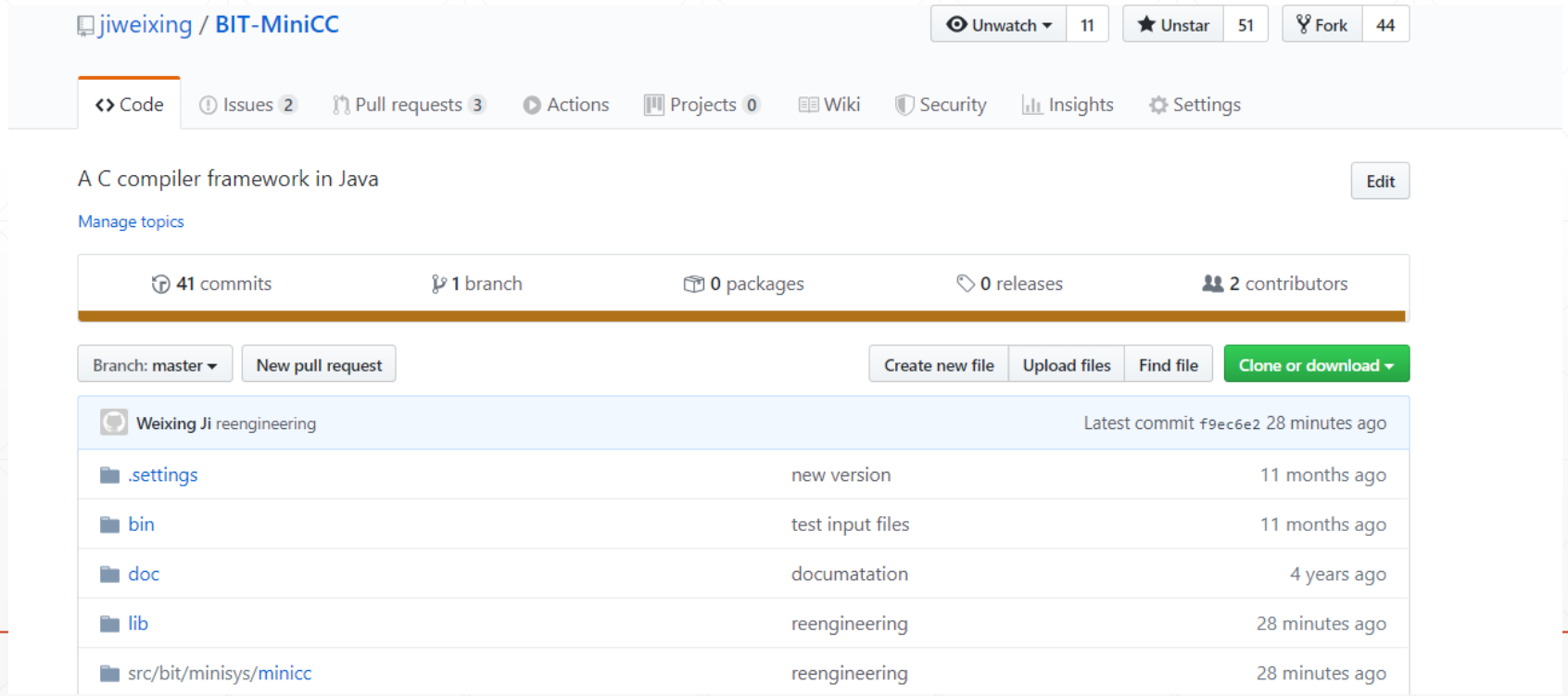
框架介绍

- 内部集成的功能
  - 预处理：注释和宏替换
  - 词法分析：所有单词
  - 语法：所有
  - 语义：正在修改
  - 中间代码：正在修改
  - 优化：暂无
  - 代码生成：正在修改

```
PROGRAM → FUNCTIONS
FUNCTIONS → FUNCTION FLIST
FLIST → FUNCTION FLIST | ε
FUNCTION → TYPE TKN_ID TKN_LP ARGS TKN_RP FUNC_BODY
ARGS → FARGS ALIST | ε
ALIST → TKN_COMMA FARGS ALIST | ε
FARGS → TYPE TKN_ID
FUNC_BODY → TKN_LB STMTS TKN_RB
STMTS → STMT STMTS | ε
STMT → EXPR_STMT | RET_STMT
EXPR_STMT → EXPR TKN_SEMICOLON
RET_STMT → TKN_KW_RET EXPR_STMT
EXPR → TERM TLIST
TLIST → TKN_PLUS TERM TLIST | ε
TERM → FACTOR FLIST
FLIST → TKN_MUL FACTOR FLIST | ε
FACTOR → TKN_LP EXPR TKN_RP | TKN_ID
TYPE → TKN_INT | TKN_FLOAT
```

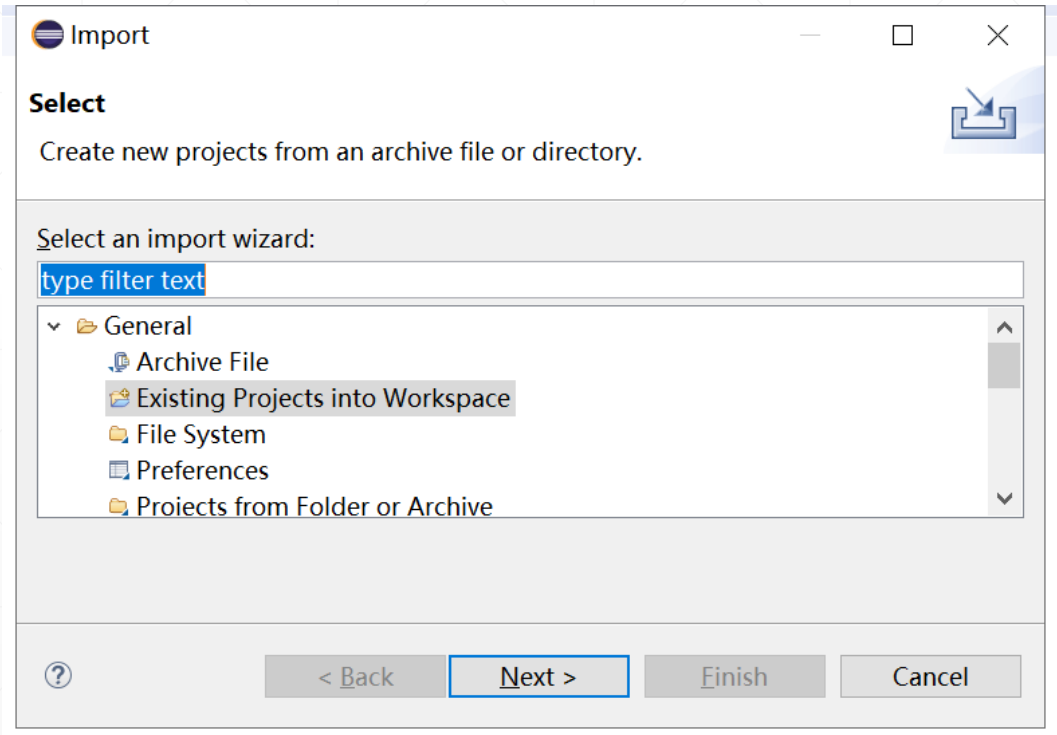
BIT-MiniCC使用方法

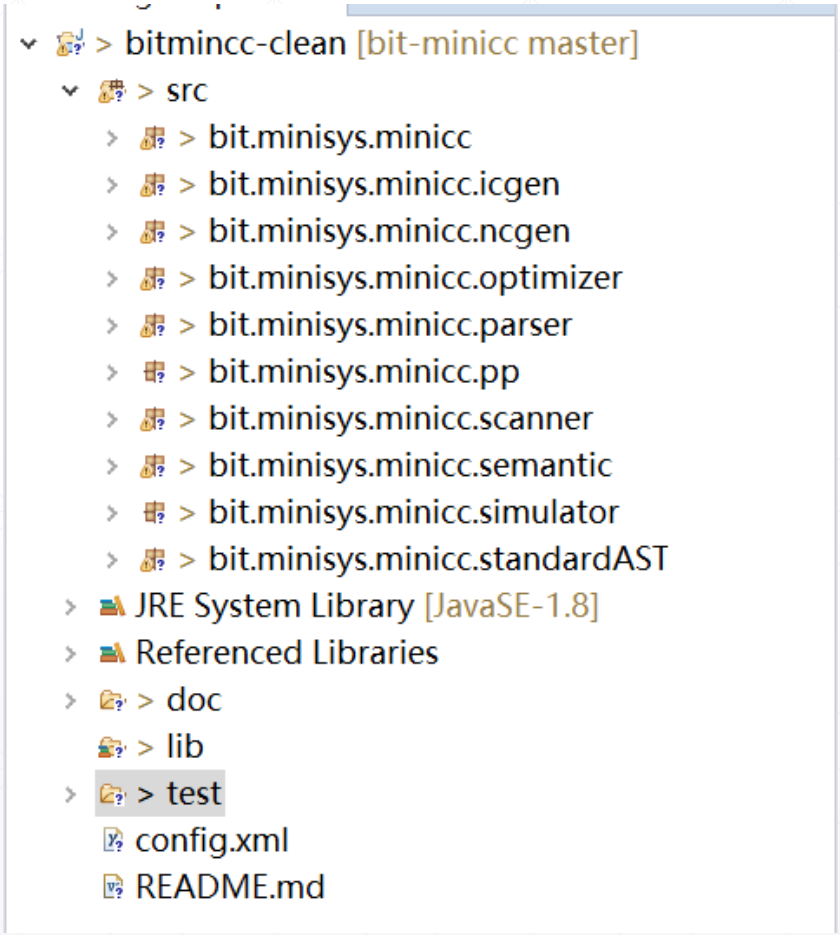
- 从github下载框架
  - https://github.com/jiweixing/BIT-MiniCC



BIT-MiniCC使用方法

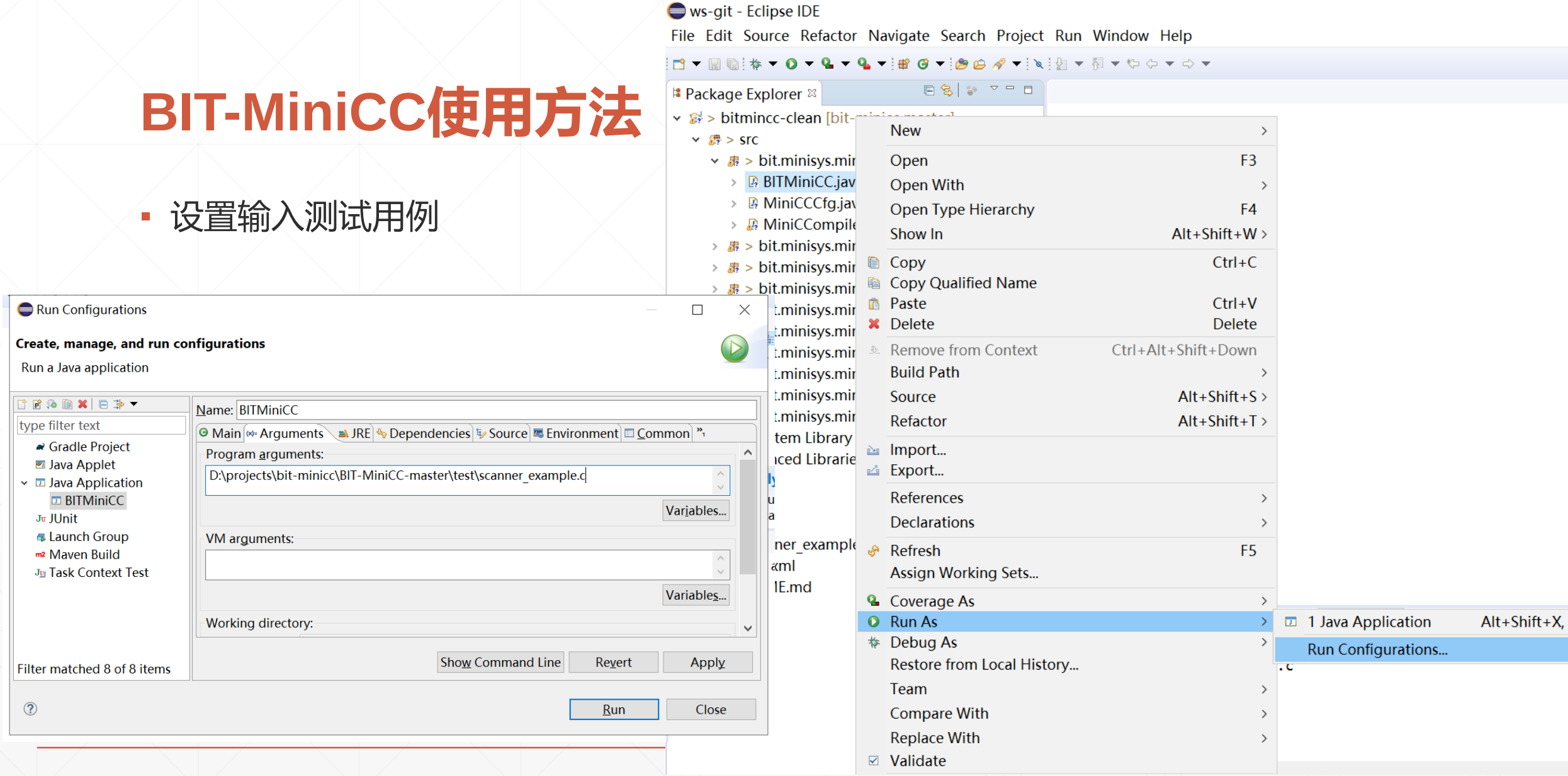
- 从Eclipse里面导入项目





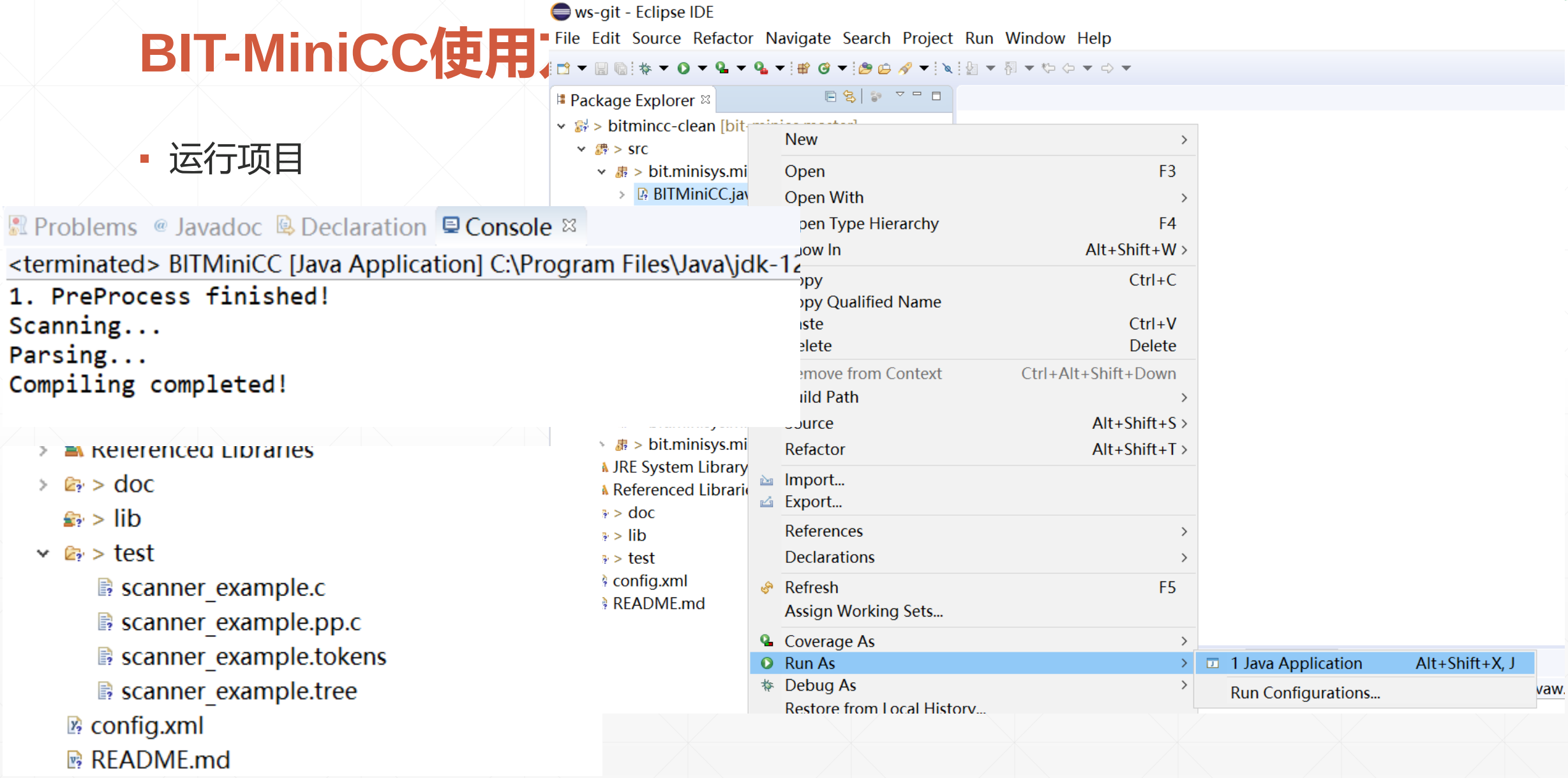
BIT-MiniCC使用方法

- 设置输入测试用例



BIT-MiniCC使用

- 运行项目

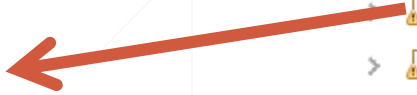


BIT-MiniCC使用方法

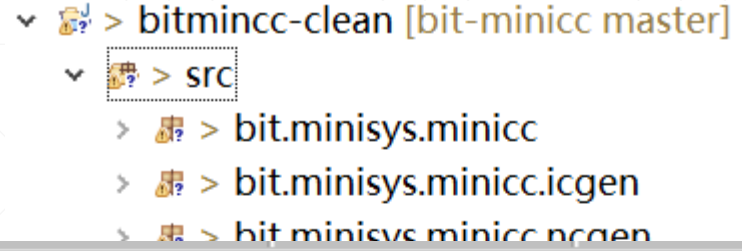
- 源代码树

ExampleScanner.java

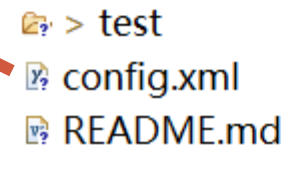
```
25 public class ExampleScanner implements IMiniCCScanner {
26
27     private int lIndex = 0;
28     private int cIndex = 0;
29
30     private ArrayList<String> srcLines;
31
32     private HashSet<String> keywordSet;
33
34     public ExampleScanner(){
35         this.keywordSet = new HashSet<String>();
36         this.keywordSet.add("int");
37         this.keywordSet.add("return");
38     }
39
40     private char getNextChar() {
41         char c = Character.MAX_VALUE;
42         while(true) {
43             if(lIndex < this.srcLines.size()) {
44                 String line = this.srcLines.get(lIndex);
45                 if(cIndex < line.length()) {
46                     c = line.charAt(cIndex);
47                     cIndex++;
48                     break;
49                 }else {
```



BIT-MiniCC使用方法

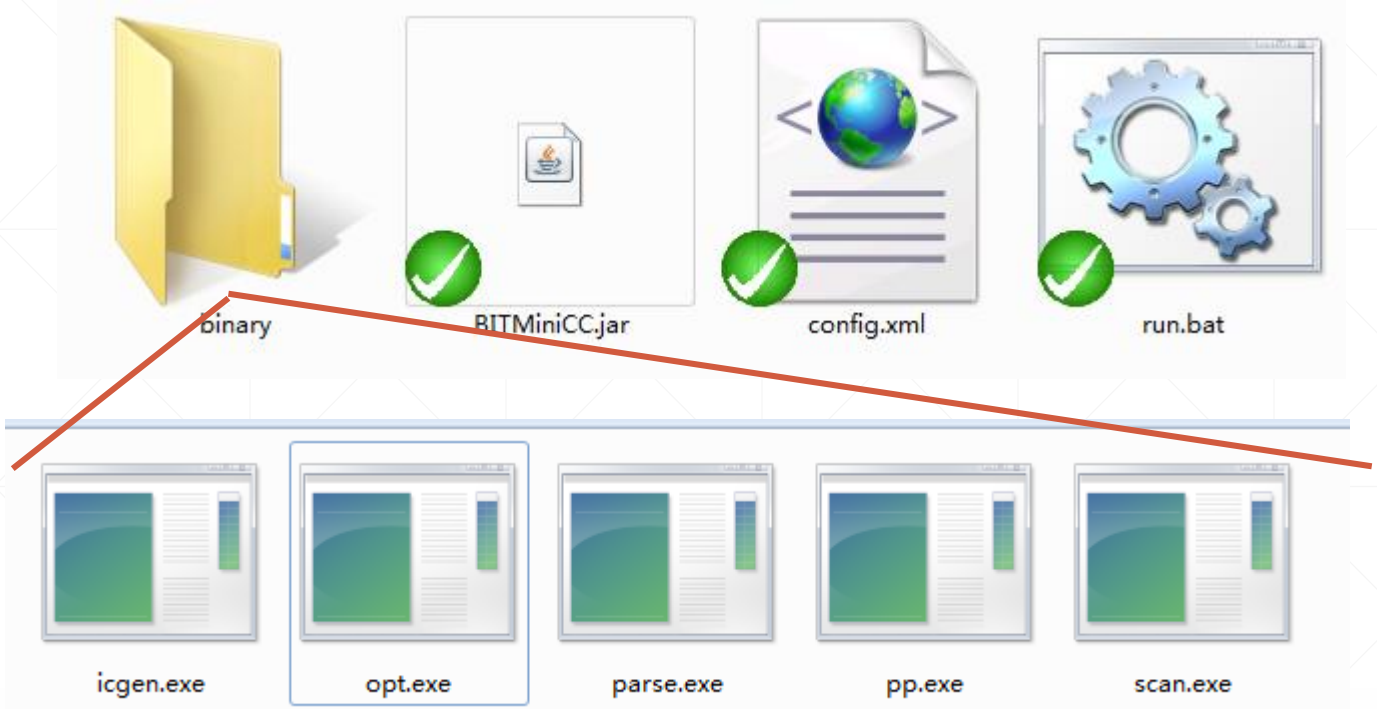



```xml
<?xml version="1.0" encoding="UTF-8"?>
<config name="config.xml">
  <phases>
    <phase>
      <phase skip="false" type="java" path="" name="preprocess" />
      <phase skip="false" type="java" path="bit.minisys.minicc.scanner.ExampleScanner" name="scan" />
      <phase skip="false" type="java" path="bit.minisys.minicc.parser.ExampleParser" name="parse" />
      <phase skip="true" type="java" path="" name="semantic" />
      <phase skip="true" type="java" path="" name="icgen" />
      <phase skip="true" type="java" path="" name="optimize" />
      <phase skip="true" type="java" path="" name="ncgen" />
      <phase skip="true" type="mips" path="" name="simulate" />
    </phase>
  </phases>
</config>
```



扩展方法

- C/C++/C#




```
<?xml version="1.0" encoding="UTF-8" ?>
- <config name="config.xml">
  - <phases>
    - <phase>
        <phase skip="false" type="binary" path="D:\projects\bit-minicc\bit-minic-clean\run\binary\pp.exe" name="pp" />
        <phase skip="false" type="binary" path="D:\projects\bit-minicc\bit-minic-clean\run\binary\scan.exe" name="scanning" />
        <phase skip="false" type="java" path="" name="parsing" />
        <phase skip="false" type="java" path="" name="semantic" />
        <phase skip="false" type="java" path="" name="icgen" />
        <phase skip="false" type="java" path="" name="optimizing" />
        <phase skip="false" type="java" path="" name="codegen" />
        <phase skip="false" type="java" path="" name="simulating" />
      </phase>
    </phases>
  </config>
```

Q & A?

jwx@bit.edu.cn