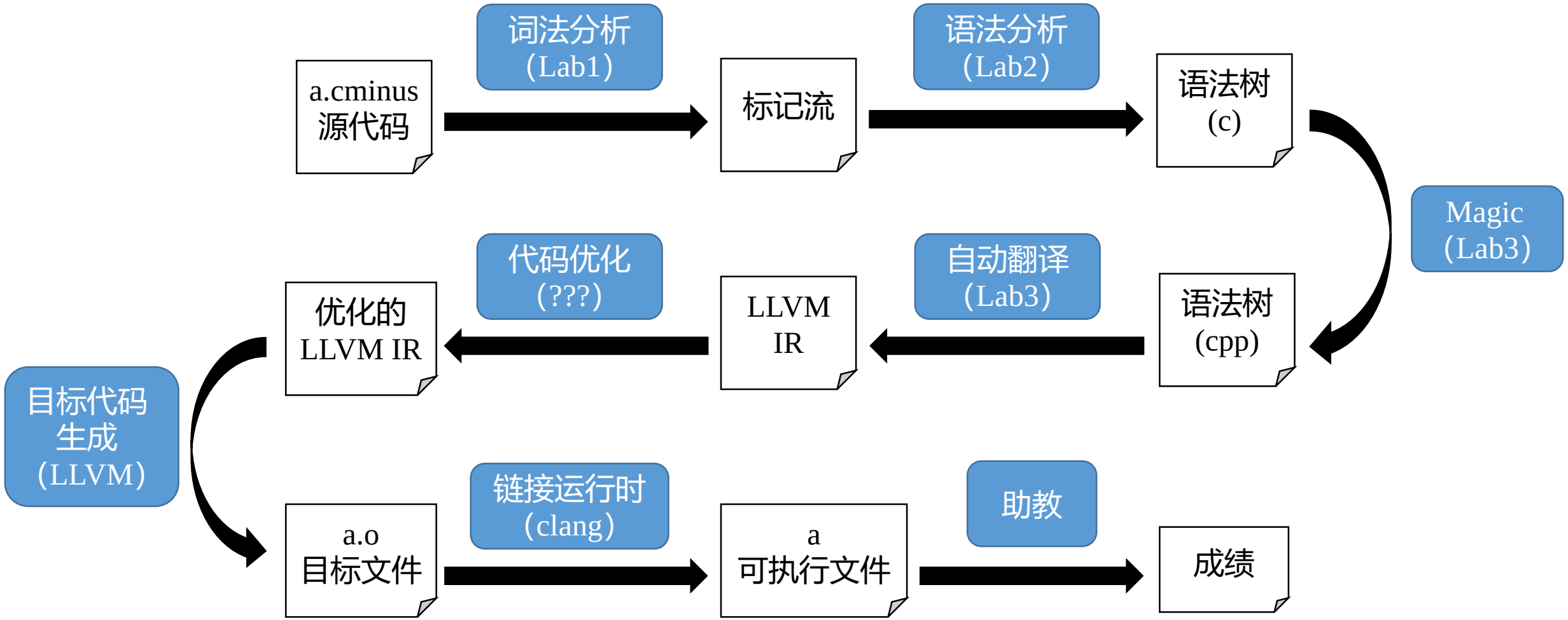
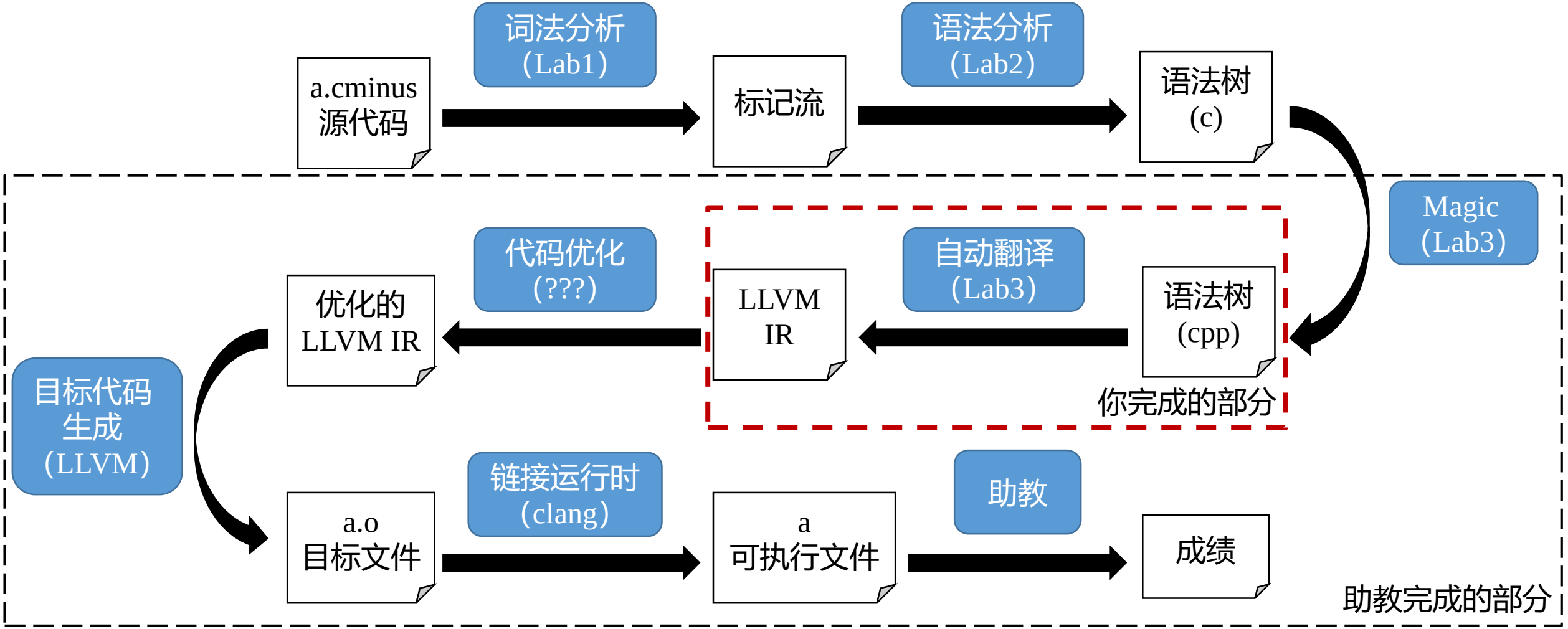
Lab3-1: C-minus程序的IR生成器

负责助教：李嘉豪 金泽文

0. 架构一览


a.cminus
源代码
词法分析
(Lab1)
标记流
语法分析
(Lab2)
语法树
(c)
Magic
(Lab3)
语法树
(cpp)
自动翻译
(Lab3)
LLVM
IR
代码优化
(???)
优化的
LLVM IR
目标代码
生成
(LLVM)
a.o
目标文件
链接运行时
(clang)
a
可执行文件
助教
成绩

0. 架构一览


a.cminus
源代码
词法分析
(Lab1)
标记流
语法分析
(Lab2)
语法树
(c)
Magic
(Lab3)
语法树
(cpp)
自动翻译
(Lab3)
LLVM
IR
你完成的部分
代码优化
(???)
优化的
LLVM IR
目标代码
生成
(LLVM)
a.o
目标文件
链接运行时
(clang)
a
可执行文件
助教
成绩
助教完成的部分

I. 基础 – 语义

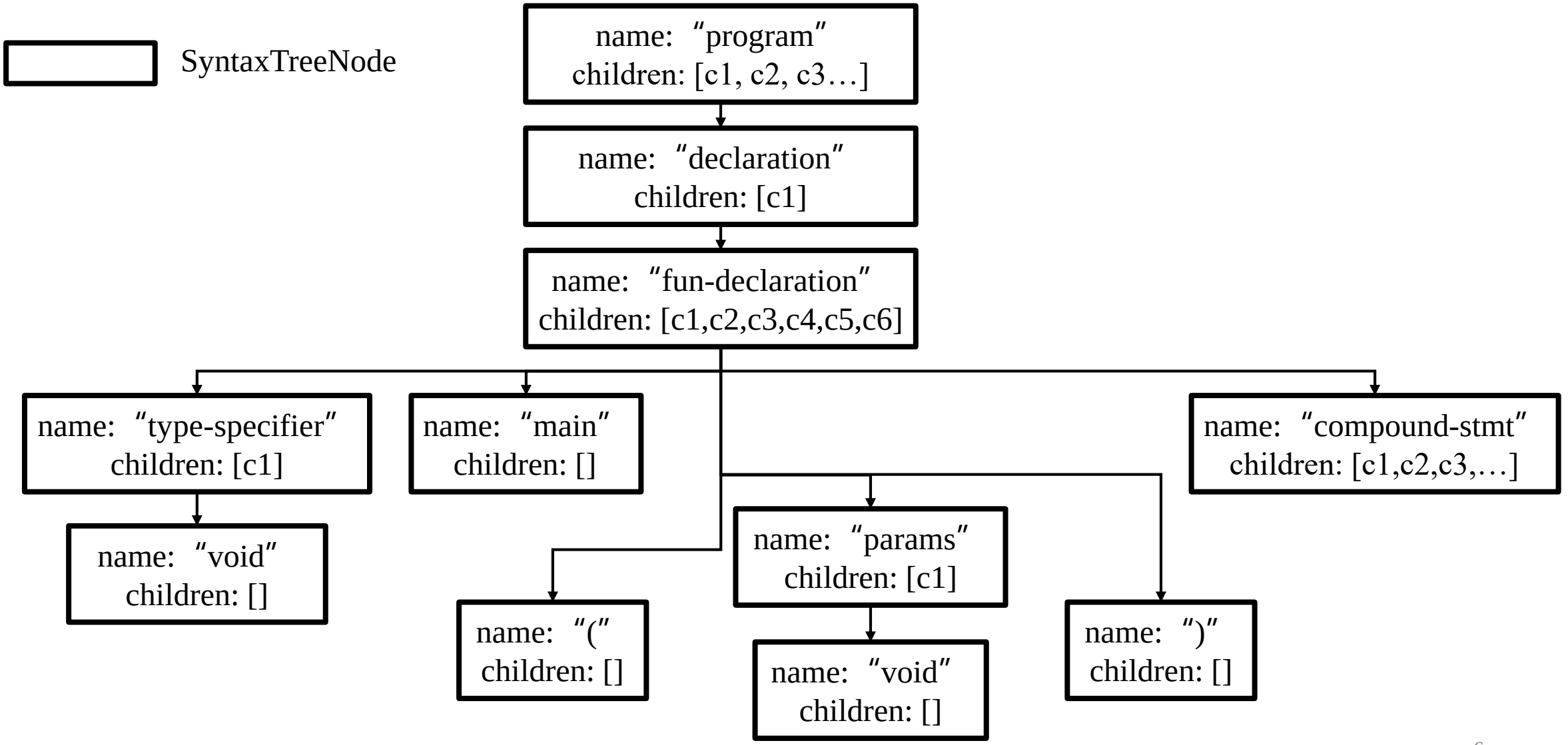
仔细阅读 cminus.md与design.md

I. 基础 – C++

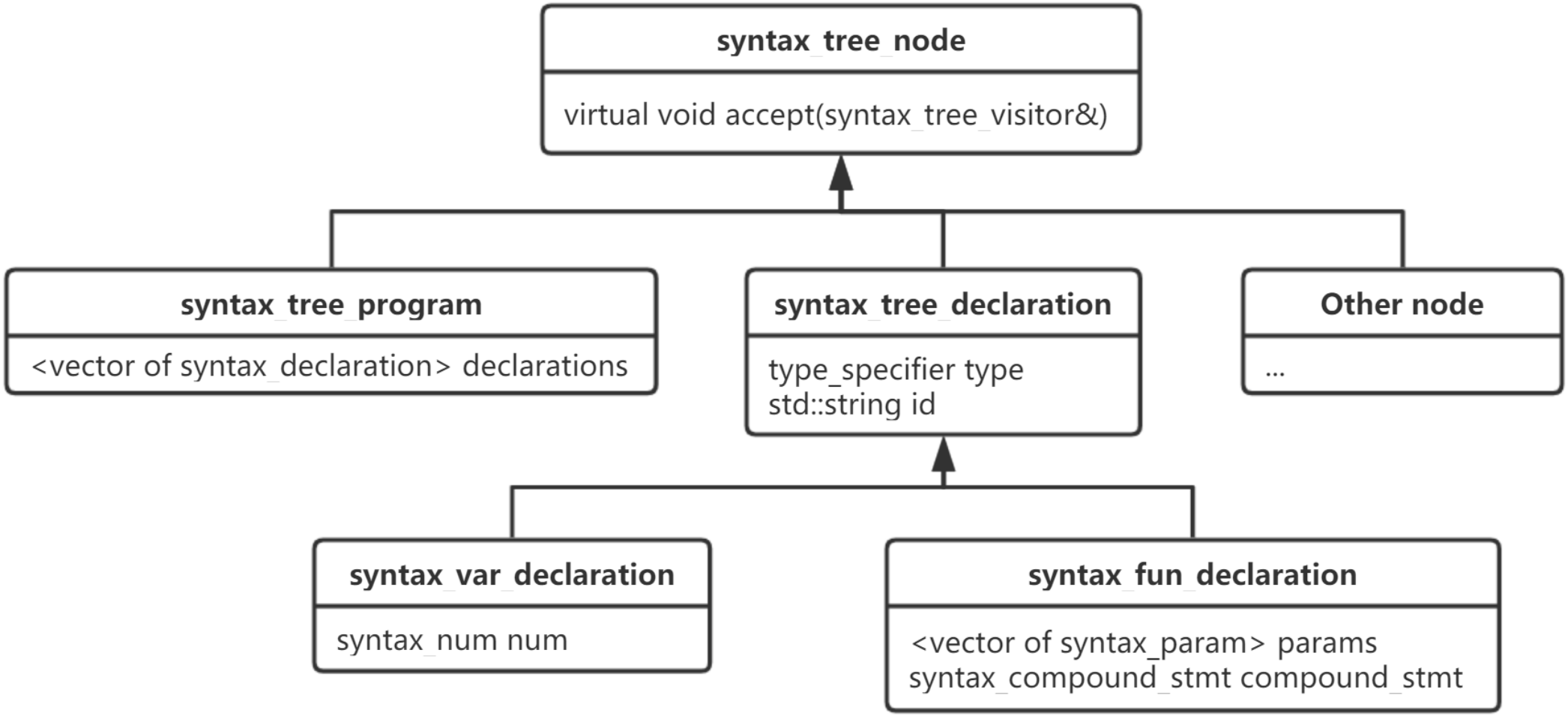
如果没有任何C++基础：

1. 阅读design.md，并动手测试（如果想要理解这个框架）
2. 参考cppreference.com

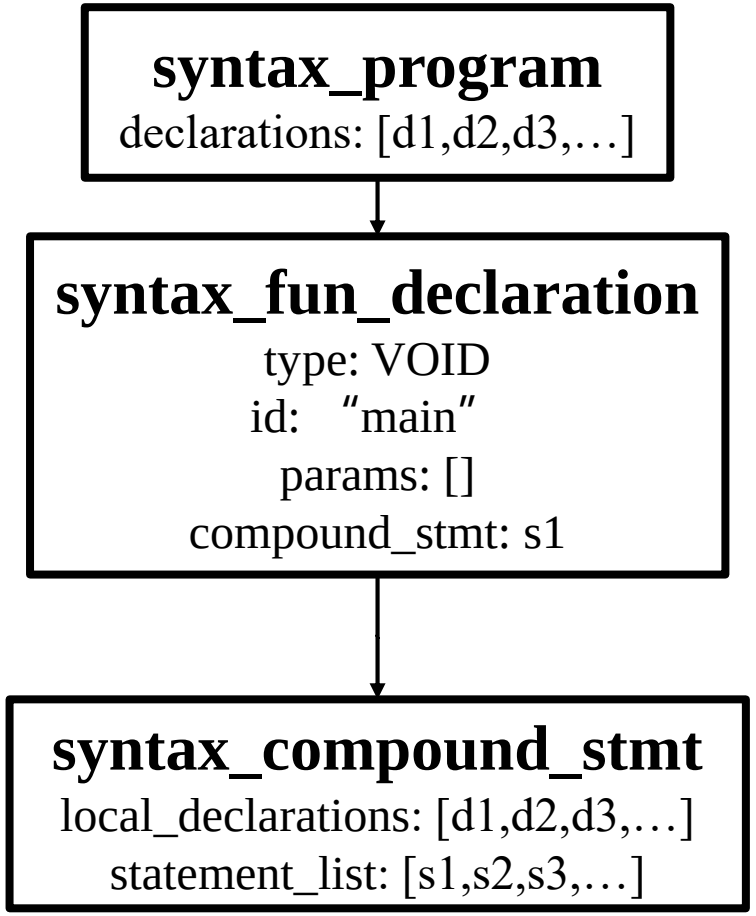
2. C++的语法树


SyntaxTreeNode
name: "program"
children: [c1, c2, c3…]
name: "declaration"
children: [c1]
name: "fun-declaration"
children: [c1,c2,c3,c4,c5,c6]
name: "type-specifier"
children: [c1]
name: "void"
children: []
name: "main"
children: []
name: "("
children: []
name: "params"
children: [c1]
name: "void"
children: []
name: ")"
children: []
name: "compound-stmt"
children: [c1,c2,c3,…]

2. C++的语法树


syntax_tree_node
virtual void accept(syntax_tree_visitor&)
syntax_tree_program
<vector of syntax_declaration> declarations
syntax_tree_declaration
type_specifier type
std::string id
Other node
...
syntax_var_declaration
syntax_num num
syntax_fun_declaration
<vector of syntax_param> params
syntax_compound_stmt compound_stmt

2. C++的语法树


syntax_program
declarations: [d1,d2,d3,…]
syntax_fun_declaration
type: VOID
id: "main"
params: []
compound_stmt: s1
syntax_compound_stmt
local_declarations: [d1,d2,d3,…]
statement_list: [s1,s2,s3,…]

3. 访问者模式

访问者模式(Visitor Pattern)是一种编程模式，该模式可以在不改变各元素的类的前提下定义作用于这些元素的新操作。实现时，在数据基础类里面有一个方法接受访问者，将自身引用传入访问者。在我们的语法树实现中，基类syntax_tree_node拥有一个虚函数accept，它就是用来接受访问者的。

3. 访问者模式

node->accept(visitor) 经过一系列调用会变成类似visitor.visit(node)的形式，而visit函数中可以定义算法内容。

简而言之，利用该模式能方便的定义遍历语法树的算法

3. 访问者模式

下面是打印语法树算法的一个visit函数的代码：

```
void syntax_tree_printer::visit(syntax_program &node) {
    _DEBUG_PRINT_N_(depth);
    std::cout << "program" << std::endl;
    add_depth();
    for (auto decl: node.declarations) {
        decl->accept(*this);
    }
    remove_depth();
}
```

3. 访问者模式

打印语法树算法的代码位于以下文件中：

include/syntax_tree.hpp

src/syntax_tree_cpp/syntax_tree.cpp

你们可以参考这个实现，以熟悉该语法树的使用

4. 更多的LLVM IR

1. 使用GlobalVariable来创建全局变量(之前的ppt有)
2. 使用CreateGEP来创建getelementptr以访问数组元素

   （使用前先阅读 http://llvm.org/docs/GetElementPtr.html）
3. 阅读IR的官方指南，了解icmp，zext，trunc等指令
4. 了解llvm::Value类的getType()和llvm::Type类的

   isIntegerTy()、isPointerTy()等函数（可能有用）

5. 实验内容

- 填充src/cminusc/cminus_builder.cpp中的16个visit函数（函数可以为空，这取决于你们怎么做）
- 定义全局变量来共享一些状态（如表达式的结果值等）
- testcase/下有一个最大公约数程序以供测试，你们可能需要自己构建测试来确保程序的可靠性（不需要提交）

5. 实验内容

你们要提交的文件（我们实际评分的文件）：

```
.
├── src
│   └── cminusc
│       └── cminus_builder.cpp // 实际的算法
└── report
    ├── contribution.md // 队长记录的队员贡献
    ├── lab3-1_report.md // 实验报告
    └── records.md // 队长记录的组队讨论与学习记录
```

5. 实验内容

实验中使用的词法分析器与语法分析器为助教实现版本的部分修改版本，如果想要使用自己的实现完成实验，可以替换deps/analyzer中的文件，但是必须自行保证接口兼容性

（助教测试时不会拷贝该目录）

6. 团队建议

- 合理安排学习计划，确保团队效率最高
- 编程任务可以按函数分配，但是需要提前沟通好事宜
- 测试可以并发进行（有人找bug，有人修bug）

Happy coding !