编译原理大作业

# 一、目标

## 1、背景知识介绍

C-- 语言是本实验的源语言。是一个 c 语言的子集，C-- 语言是单文件的，去除了 C 语言中的 include/define/pointer/struct 等较复杂特性。

LLVM 是一个模块化的、可重用的编译器和工具链的集合，目的是提供一个现代的、基于 SSA 的、能够支持任意静态和动态编译的编程语言的编译策略。在最近几年已经成为表现上能够和 gcc 对标的项目。

LLVM IR 是 LLVM 项目中通用的中间代码，作为源语言和体系架构的连接部分，是学生需要从源语言中编译并翻译到的目标语言。

## 2、大作业要求

本次大作业要求编写一个编译器前端（包括词法分析器、语法分析器、语义分析及中间代码生成），（1）【必做】使用自动机理论编写词法分析器，（2）【必做】自上而下或者自下而上的语法分析方法编写语法分析器。(3) 【选做】补充完成中间代码生成部分代码。

1、**【必做】**编写C--语言的词法分析器，理解词法分析器的工作原理，熟练掌握基于自动机理论的词法分析器的工作流程。编写源代码识别输出单词的二元属性，填写符号表。

2、**【必做】**编写C--语言的语法分析器，理解自上而下/自下而上的语法分析算法的工作原理；理解词法分析与语法分析之间的关系。语法分析器的输入为 C--语言源代码， 输出为按扫描顺序进行推导或归约的正确/错误的判别结果，以及按照最左推导顺序/规范规约顺序生成语法树所用的产生式序列。

3、**【选做】**补全给出的中间代码生成部分代码。将编译器的前端与我们提供的编译器中端衔接，该部分需要遍历语法分析器生成的语法树，访问语法树结点并调用我们所提供的中端代码(见附录 2.2)，最终输出中间代码。

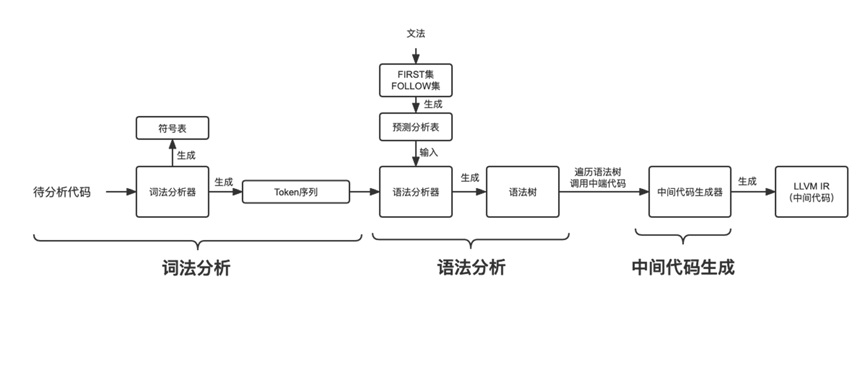


图 1: 整体流程图

# 二、软件需求

本次大作业要求实现的词法分析器和语法分析器可以**组成一个完整的编译流程**，即针对输入的C--语言，可以按照上面的流程图，给出词法分析器的单词符号序列和语法分析器的产生式序列。选做任务不要求融合进编译流程。

## 1、词法分析器（必做）

1. 完成 C--语言的词法分析器，要求采用课程教授方法，实现有限自动机**确定化，最小化**算法。词法分析器的输入为 C--语言源代码，输出识别出单词的二元属性，填写符号表。单词符号的类型及具体要求如下：

**关键字（KW，不区分大小写）包括：** (1) int (2) void (3) return (4) const

**运算符（OP）包括：**(5) + (6) - (7) \* (8) / (9) % (10) = (11) > (12) < (13) ==

(14) <= (15) >= (16) != (17) && (18) ||

**界符（SE）包括：**(19) ( (20) ) (21) { (22) } (23)； (24) ,

**标识符（IDN）**定义与 C 语言保持相同，为字母、数字和下划线（\_）组成的不以数字开头的串。为方便起见，本次作业中定义“main”也属于标识符。标识符的正则表达式为：IDN=[a‐zA‐Z\_][a‐zA‐Z\_0‐9]\*

**整数（INT）**定义与 C 语言类似。为方便起见，本次作业中定义整数包含负数。整数的正则表达式为INT=-\*[0‐9]+

1. 实现语言：C/C++/Java/Python。注意：中端代码为 C++编写，若决定做选做部分， 推荐首选 C++。
2. 操作目的：生成符号表；将源代码转化为单词符号序列。
3. 输出：请严格按照第三章“输出示例”的**格式要求**，将结果保存为文本文件（.txt）。

## 2、语法分析器（必做）

1. 完成 C--语言的语法分析器，语法分析器的输入为 C--语言代码的单词符号序列（即语法分析器的输出），输出用最左推导或规范规约产生语法树所用的**产生式序列**。C--语言文法会包含以下操作：

① 常量定义的声明（int a;）

② 常量定义的初始化（int a = 3;）

③ 常类型数据的声明和定义（const int a = 3;）

④ 变量赋值传递（已声明的情况）（a = 3;/a = b;）

⑤ 函数名称，返回值类型，函数内部建立初始基本块并插入返回指令（int add(...)）

⑥ 函数传入参数的创建，管理（int add(int a, int b)）

⑦ 一元运算表达式（加法和乘法），（int a = 3 + 4; int c = a \* b;）

⑧ 单目运算符号和变量数值结合（+/-/!）（!0/ -3）

⑨ 复合运算表达式（多个一元表达式结合 a = (b + c) \* d）

⑩ 大小比较（ >/ >=/ </ <=）

⑪ 双目运算符（|| && <= == !=）

**注意：**语法分析使用本次作业提供的文法，即文件“grammar.txt”。文法规则格式如下：

### [产生式头部] [空格] -> [空格] [产生式体]

注：文法中出现的$代表ε。文法中出现的 IDN（标识符）、INT（整数）参照词法分析器中定义。所有的关键字、界符、运算符均以原本形式存在，即终结符就是词法分析器中定义的形式。

1. 流程说明：以 LL（1）语法分析方法为例，构建语法分析器需要完成以下内容：
   * 1. 求FIRST 集合、FOLLOW 集合
     2. 匹配对待编译代码规约使用的产生式序列。
2. 实现语言：C/C++/Java/Python. 注意：中端代码为 C++编写，若决定做选做部分， 推荐首选 C++。
3. 输出：请严格按照第三章“输出示例”的格式要求，将结果保存为文本文件（.txt）。

## 3、中间代码生成（选做）

1. LLVM IR 具有三种表示形式，这三种中间格式是完全等价的：
2. : 在内存中的编译中间语言（我们无法通过文件的形式得到）
3. : 在硬盘上存储的二进制中间语言（格式为 .bc）
4. : 人类可读的代码语言（格式为 .ll） 本次实验要求输出 .ll 形式的 LLVM IR。

(2) 中间代码生成器代码补全：对给出的中间代码生成器部分代码的缺失部分，即各个结点的 visitor 函数进行补全。针对每个结点构造一个 visitor 函数来调用中端代码类(见附录1.2)，**其中 visit 函数对应于图一中语法分析和中间代码生成的衔接部分。**

//varDef : bType Ident '=' initVal #initVarDef

//本例子展示一个int 类型全局变量声明+初始化（例：int a = 10 ; ）的中间代码生成过程的伪代码

//对变量类型，变量名和变量取值的初始化

1. Type type = i32\_Type;
2. string var\_name = “”;
3. int data = 0;

//InitVarDefContext \*ctx 是AST 的对应结点

1. visitInitVarDef(InitVarDefContext \*ctx) {

//访问子结点

1. visitChildren(ctx);

//s 为符号表，其中s.variable()可以调用该结构存储变量

1. map<string, Value\*> VariableCollection = s.variable();

//end()函数表示遍历完变量的集合，没有找到相同命名的变量

1. if(VariableCollection.find(var\_name) != VariableCollection.end()) cout << " 重复命名" << endl;

//判断该变量是否为全局变量

1. if(ctx.isGlobal){

//调用中端的GlobalVarible 函数，生成该全局变量声明和初始化过程的中间代码

//参数说明：var\_name(变量名),m(模块，一个源代码文件的抽象表达，参考附录 2.2 中的Module 类),i32\_Type(数据类型),false(判定是否是常量变量,是否有const 关键字), ConstantInt::get(data, m)(将初始化的数据存储)

1. GlobalVariable\*gv=GlobalVariable::create(var\_name,m,i32\_Type,false, ConstantInt::get(data, m));

//将变量名和中间代码存储到符号表s 中

1. s.put(var\_name, gv); 11 }

12 return nullptr;

13 }

**代码 1 VarDef Visitor 函数(部分)伪代码示例（全局变量声明和初始化）**

1. visitChildren(ParseTree \*ctx) {
2. for(int i=0;i< (int)ctx‐>children.size(); i++){

//children.size()得到ctx 节点的子节点数

//访问 bType 子结点获取变量类型

1. if(ctx‐>children[i]‐>type == “bType”){

4

visitBType(ctx‐>children[i]);

5 }

//访问 ident 子结点获取变量名

1. else if(ctx‐>children[i]‐>type == “Ident”){
2. visitIdent(ctx‐>children[i]);

//访问 initVal 子结点获取变量的值，该过程(根据文法)涉及层层调用，为了方便理解直接写为访问存放数据的

Number 结点

1. }else if(ctx‐>children[i]‐>type == “initVal”){

9

visitNumber(ctx‐>children[i]);

10 }

11 }return nullptr; 12 }

1. visitBType(ParseTree \*ctx) {
2. type = ctx‐>getText(); 15 }
3. visitIdent(ParseTree \*ctx) {
4. var\_name = ctx‐>getText(); 18 }
5. visitNumber(NumberContext \*ctx) {
6. data = ctx‐>getText();

21 }

**代码 2 visit 子结点代码示例**

（3）实现语言：C++；

（4）操作目的：完成对语法分析生成的抽象语法树的遍历，并通过访问语法树结点来调用我们提供的中端代码（见附录 1.2），生成 LLVM IR 中间代码。

# 三、输出示例

## 1、词法分析输出示例

int a = 10; int main(){

a=10;

return 0;

}

**代码 3 待测 C--代码**

（1）**【必须按规定格式输出】**输出单词符号序列：

**输出格式：将完整输出严格按以下格式存入txt文件中**

### [待测代码中的单词符号] [TAB] <[单词符号种别] >

其中，单词符号种别为 KW（关键字）、OP（运算符）、SE（界符）、IDN（标识符）INT（整形数）；单词符号内容第一个维度为其种别，第二个维度为其属性。[TAB]为一个制表符‘\t’

int <KW>

a <IDN>

= <OP>

10 <INT>

; <SE>

int <KW>

main <IDN>

( <SE>

) <SE>

{ <SE>

a <IDN>

= <OP>

10 <INT>

; <SE>

return <KW>

0 <INT>

; <SE>

} <SE>

**代码 4 单词符号序列示例**

## 2、语法分析输出示例（以预测分析为例）

1. **【必须按规定格式输出】**输出规约序列（此处仅展示一部分）：

### 输出格式：将完整结果严格按以下格式存入txt文件中

**[序号] [TAB] [栈顶符号]#[面临输入符号] [TAB] [执行动作]**

选用文法规则为“grammar.txt”所示；执行动作为“reduction”（LL 中的推到或 LR 中的归约），“move”（LL 分析的跳过或 LR 分析的移进），“accept”（符合文法规则接受）或“error”（不符合文法规则拒绝）。注：栈空用“EOF”表示。

1. program#int reduction
2. compUnit#int reduction
3. decl#int reduction
4. valDecl#int reduction
5. btype#int reduction
6. int#int move
7. varDef#IDN reduction

…………

85 }#} move

86 compUnit#EOF reduction

87 EOF#EOF accept

**代码 5 归约序列部分输出示例**

注：分析栈左端为栈顶，输入串过长没有在输出实例中进行展示，输入串即词法分析器生成的单词符号序列。

## 3、中间代码输出示例

; ModuleID = 'sysy2022\_compiler' source\_filename = "../input/01\_var\_defn.sy" @a = global i32 0

declare i32 @getint() declare i32 @getch() declare i32 @getarray(i32\*) declare void @putint(i32) declare void @putch(i32)

declare void @putarray(i32, i32\*) declare void @starttime() declare void @stoptime()

define i32 @main() { main\_ENTRY:

%op0 = load i32, i32\* @a store i32 %op0, i32 10

ret i32 0

}

**代码 6 中间代码输出示例**

# 四、提交要求

## 1、源代码

包括词法分析器、语法分析器、对中端调用的部分

## 2、开发报告

对项目的开发过程进行详细的描述，包括（1）词法分析器算法描述，输出格式说明， 源程序编译步骤；（2）语法分析器的算法描述，创建的分析表（预测分析表、LR 分析表等），输出格式说明，源程序编译步骤。（3）存储，遍历语法树的过程算法伪代码，以及调用中端的过程的算法设计思想。

# 五、注意事项：

* 1. 采用文件中给定的文法规则（grammar.txt）编写程序；作业结果将通过后台代码测试正确性，请严格按要求格式输出结果。
  2. 提供的文件中给出的5个测试用例及要求的输出结果，供同学们测试自己的程序用。
  3. 不得借助 Lex/Yacc，ANTLR 等编译器自动生成工具。
  4. 所有出现在报告、代码、展示 PPT 中的内容，若非原创，需要明确标明内容来源，并给出正规的引用。未标明引用而采纳他人内容的情况，**视为抄袭**，遵照天津大学相关条例处理。
  5. 小组内各位组员需合理分工配合完成大作业，每位组员对项目的贡献可明确区分。 PPT 展示时需明确每位组员负责完成的工作内容。

# 附录

## 1、LLVM 相关内容

**1.1 LLVM 语法格式介绍：**

https://llvm.org/docs/LangRef.html

## 1.2 中端代码链接：

https://gitee.com/happy-traveller/compiler