Exercise Sheet

09/26

1. 【判断题】基于下面语法（利用上课所学算法）进行移入规约分析时是否可能产生移入 /规约冲突，其中`E`与`L`是非终结符，其他符号都是终结符。

```
L -> E          E -> ID ( )          E -> NAT
L -> L , E      E -> ID ( L )        E -> ID
                E -> ( TYPE_NAME ) E
```

2. 请先指出基于下面语法（利用上课所学算法）进行移入规约分析时可能产生的两类移入 /规约冲突，再设计基于优先级与结合性的规则解决这些冲突，其中`E`是非终结符，其他符号都是终结符。

```
E -> E + E      E -> E [ E ]      E -> ID
```

3. 请先指出基于下面语法（利用上课所学算法）进行移入规约分析时可能产生的移入/规约冲突，再设计基于优先级与结合性的规则解决这些冲突，其中`S`与`L`是非终结符，其他符号都是终结符。

```
S -> IF E THEN S ELSE S     S -> SIMPLE ;     L -> S
S -> IF E THEN S            S -> { L }        L -> S L
                            S -> { }
```

4. 考虑 C 语言中 struct/union/enum 的定义与声明，基于 typedef 的类型定义，以及变量的定义。下面是它们的语法（本题中不需要考虑一条语句定义多个变量的情形，也不需要考虑变量定义同时初始化的情形）：

```
STRUCT_DEFINITION ::= struct STRUCT_NAME { FIELD_LIST } ;
STRUCT_DECLARATION ::= struct STRUCT_NAME ;
UNION_DEFINITION ::= union UNION_NAME { FIELD_LIST } ;
UNION_DECLARATION ::= union UNION_NAME ;
ENUM_DEFINITION ::= enum ENUM_NAME { ENUM_ELE_LIST } ;
ENUM_DECLARATION ::= enum ENUM_NAME ;
TYPE_DEFINITION ::= typedef LEFT_TYPE NAMED_RIGHT_TYPE_EXPR ;
VAR_DEFINITION ::= LEFT_TYPE NAMED_RIGHT_TYPE_EXPR ;
```

本题约定，struct 与 union 的域列表允许为空，enum 的元素列表不得为空。

```
FIELD ::= LEFT_TYPE NAMED_RIGHT_TYPE_EXPR ;
FIELD_LIST ::= FIELD FIELD ... FIELD
ENUM_ELE_LIST ::= ENUM_ELE, ENUM_ELE, ... , ENUM_ELE
```

这里提到的`STRUCT_NAME`、`UNION_NAME`、`ENUM_NAME`、`ENUM_ELE`以及下面会提到的`IDENT`（标识符）都表示以字母或下滑线开头且仅包含字母数码与下划线的名字。需要特别注意的是，C 语言的中变量定义与域定义中，变量类型与域类型都是通过两部分进行描述的：左半部分是基础类型，右半部分是包含变量名或域名的一个表达式。例如，`int * x`这个定义可以分为`int`与`* x`两个部分，它表示`* x`的值（即存储在`x`地址的内容）为整数类型。这就是上面提到的：

`LEFT_TYPE` `NAMED_RIGHT_TYPE_EXPR`

本题中需要考虑指针类型、数组类型、函数类型的情形，在基础类型方面只考虑 `int` 与 `char` 两个类型：

```
LEFT_TYPE ::= struct STRUCT_NAME { FIELD_LIST }
            | struct { FIELD_LIST }
            | struct STRUCT_NAME
            | union UNION_NAME { FIELD_LIST }
            | union { FIELD_LIST }
            | union UNION_NAME
            | enum ENUM_NAME { ENUM_ELE_LIST }
            | enum { ENUM_ELE_LIST }
            | enum ENUM_NAME
            | int | char | IDENT
```

```
NAMED_RIGHT_TYPE_EXPR ::= IDENT
                        | * NAMED_RIGHT_TYPE_EXPR
                        | NAMED_RIGHT_TYPE_EXPR [ NAT ]
                        | NAMED_RIGHT_TYPE_EXPR ( ARGUMENT_TYPE_LIST )
ANNON_RIGHT_TYPE_EXPR ::= EMPTY
                        | * ANNON_RIGHT_TYPE_EXPR
                        | ANNON_RIGHT_TYPE_EXPR [ NAT ]
                        | ANNON_RIGHT_TYPE_EXPR ( ARGUMENT_TYPE_LIST )
ARGUMENT_TYPE ::= LEFT_TYPE ANNON_RIGHT_TYPE_EXPR
ARGUMENT_TYPE_LIST ::= ARGUMENT_TYPE, ..., ARGUMENT_TYPE
```

在 C 语言中，表达式（这里提到的 `NAMED_RIGHT_TYPE_EXPR` 与 `ANNON_RIGHT_TYPE_EXPR` ）后缀（数组与函数）的结合优先级高于前缀的结合优先级，并且允许使用圆括号改变优先级。例如，`int * x[10]` 表示 `int * (x[10])` ，定义了一个整数指针的数组；函数参数类型的语法也是类似的，例如 `int * [10]` 表示整数指针的数组类型， `int ( * )(int)` 表示整数一元函数的函数指针类型。本题约定，函数的参数类型列表可以为空。

这一题中，用于所有定义与声明的抽象语法树的 C 语言存储结构以及辅助构造函数、调试函数已经在 lang.h 与 lang.c 中提供了，main.c 程序也是固定的（详见 `struct_union_enum.zip`）。请编写 lang.l 与 lang.y 实现词法分析与语法分析。请记住：你设计的语法分析规则不应产生移入/规约冲突或规约/规约冲突，请善用 Bison 工具与其生成的信息 parser.output 调试并消除冲突。