词法分析（一周）

实验一 预处理

- 【任务性质】必做任务，分值10分。
- 【任务介绍】在词法分析之前，对程序员提交的源程序进行预处理，剔除注释等不必要的字符，以简化词法分析的处理
- 【输入】字符串形式的源程序。
- 【输出】处理之后的字符串形式的源程序。
- 【题目】设计一个程序，从任意字符串中剔除C语言形式的注释，包括：
    1. 形如： //...的单行注释；
    2. 形如： /*...*/ 的多行注释。

实验二 标识符的识别

- 【任务性质】必做任务，分值10分。
- 【任务介绍】根据给定源语言的构词规则，从任意字符串中识别出所有的合法标识符。
- 【输入】字符串。
- 【输出】单词符号流，一行一个单词。
- 【题目】设计一个程序，从任意字符串中识别出所有可视为C语言“名字”的子串。注意：
    1. 构词规则：以字母打头，后跟任意多个字母、数字的单词；长度不超过20；不区分大小写；把下划线视为第27个字母。
    2. 关键字保留，即：语言定义中保留了某些单词用作关键字，程序员不可以将这些单词用作“名字“（变量名、常量名、函数名、标号名等等）。

实验三 词法分析

- 【任务性质】必做任务，分值10分。
- 【任务介绍】根据给定源语言的构词规则，从任意字符串中识别出该语言所有的合法的单词符号，并以等长的二元组形式输出。
- 【输入】字符串形式的源程序。
- 【输出】单词符号所构成的串（流），单词以等长的二元组形式呈现。
- 【题目】设计一个程序，根据给定源语言的构词规则，从任意字符串中识别出该语言所有的合法的单词符号，并以等长的二元组形式输出。注意：
    1. 附录A中介绍了一个基于C语法规则设计的源语言LittleC和一个基于Pascal语法规则设计的源语言LittleP，可以作为参考。
    2. 学生可以自行挑选或设计一种源语言，以此为基础来完成本实验和后续实验。该语言的设计应该满足附录B的要求。
    3. 该程序应该设计为至少包含2个模块：驱动模块和工作模块。驱动模块包含了程序的入口和出口，主要负责输入、输出处理并调用工作模块；工作模块负责具体的分割、识别、归类等工作。这样做的好处是：只要模块间的接口（需要传递哪些数据，数据的结构）设计合理，后续实验中做语法分析器时就可以直接调用此处的工作模块，而不需要改动太多代码。

语法分析（三周）

实验四 对算术表达式的递归下降分析

- 【任务性质】必做任务，分值10分。
- 【任务介绍】根据给定的上下文无关文法，分析任意一个算术表达式的语法结构。
- 【输入】任意的算术表达式。
- 【输出】如果判断正确，则输出表达式的后续遍历，否则输出错误。
- 【题目】设计一个程序，根据给定的上下文无关文法，能够判断。要求：
    1. 基础文法：

```
<Expr> → <Term> <Expr1>
<Expr1> → <AddOp> <Term> <Expr1> | empty
<Term> → <Factor> <Term1>
<Term1> → <MulOp> <Factor> <Term1> | empty
<Factor> → id |number | ( <Expr> )
<AddOp> → + | -
<MulOp> → * | /
```

    2. 语法分析方法采用递归子程序法。
    3. 输入：形如

$$a+b*2/4-(b+c)*3$$

    的算术表达式，有+、-、*、/ 四种运算符，运算符的优先级、结合规则和括号的用法遵循惯例，有变量、整数两种运算对象。为简化问题，变量和整数均为只含有1个字符的单词，忽略 空格等非必要的字符。
    4. 输出：输入正确时，输出表达式的后续遍历；输入错误时，输出error。

实验五 对多条执行语句的递归下降分析

- 【任务性质】必做任务，分值10分。
- 【任务介绍】根据给定的上下文无关文法，对高级程序设计语言中常见的几种执行语句进行语法分析。
- 【输入】一串执行语句，其中包括：赋值语句、选择语句和循环语句。
- 【输出】如果分析正确，则输出为正确，否则输出错误。
- 【题目】设计一个程序，根据给定的上下文无关文法，对于输入的一串源程序语句，判读语法是否正确或者报告错误。要求：
    1. 基础文法以<Block>为开始符号：

```
<Block> → { <Decls> <Statements> }
<Decls> → <Decls> <Decl> | empty
<Decl> → <Type> <Names> ;
<Names> → <Names> , <Name> | <Name>
<Type> → int
<Name> → id
<Statements>的定义参照附录A中的文法定义；
```

2. 语法分析方法采用递归子程序法。
3. 输入：一串（3~5句）执行语句，其中包括：赋值语句、选择语句和循环语句。
4. 输出：如果分析正确，则输出为正确，否则输出错误。
5. 赋值语句：左部为1个简单变量（假设都定义为整型），右部为1个算术表达式；可以调用实验四中的程序来完成对这个算术表达式的分析。
6. 选择语句：包含if-then单分支和if-then-else双分支两种结构。只考虑分支判定条件为1个简单的关系运算表达式的情况，暂不处理逻辑运算。
7. 循环语句：仅包含while-do。

实验六 对完整程序的递归下降分析

- 【任务性质】必做任务，分值10分。
- 【任务介绍】递归下降的语法分析。
- 【输入】一个完整的源程序。
- 【输出】正确或者错误。
- 【题目】设计一个程序，输入字符串形式的源程序，输出该程序的是否符合相关语法，有错误时报告错 误。要求：
  1. 源语言及其语法规则：可以参照附录A，也可以自定义。
  2. 输入为字符串形式的源程序，因此，需要调用前面实验做过的词法分析器，为语法分析器提供单词符号。
  3. 应该指出错误的具体位置，如：在xx单词之后/之前发现错误，分析中止。

代码翻译（三周）

实验七 对算术表达式构造递归下降翻译器

- 【任务性质】必做任务，分值10分。
- 【任务介绍】对算术表达式做递归下降分析，同时将其翻译为中间代码。
- 【输入】算术表达式。
- 【输出】四元式序列。
- 【题目】对实验四的程序进行升级改造，使得程序对于输入的任意一个算术表达式，在对其做递归下降分析的同时，生成等价的中间代码，一遍完成。要求：
  1. 基础文法：同实验四。

2. 语法分析：沿用实验四的程序框架。
3. 语义处理：生成四元式序列。
4. 一遍处理：把语义处理的代码插入到语法分析的代码中。设计要点有两个：
    1. 语法分析走到哪里的时候应该执行语义动作；
    2. 语义动作（这里指的是生成四元式）应该怎么做。
5. 为简化问题，不考虑输入有错误的情况，不考虑语义检查。

实验八 对多条执行语句构造递归下降翻译器

- 【任务性质】必做任务，分值10分。
- 【任务介绍】对能处理多条执行语句的递归下降分析器进行改造，使其能够一遍处理，同时完成语法分析和中间代码翻译。
- 【输入】一串语句组成的语句块，其中包括：赋值语句、选择语句和循环语句。
- 【输出】与输入对应的一个四元式序列。
- 【题目】对实验五的程序进行升级改造，使得程序对于输入的任意一串语句，在对其做递归下降分析的同时，生成等价的四元式序列，一遍完成。要求：
    1. 基础文法：同实验五
    2. 语法分析：沿用实验五的程序框架，去掉生成语法树的部分。
    3. 语义处理：生成四元式序列。
    4. 一遍处理：把语义处理的代码插入到语法分析的代码中。
    5. 为简化问题，不考虑输入有错误的情况，不考虑语义检查。

实验九 构造能处理完整程序的递归下降翻译器

- 【任务性质】必做任务，分值10分。
- 【任务介绍】对递归下降分析器进行改造，使其能够一遍处理，同时完成语法分析和中间代码翻译。
- 【输入】一个完整的源程序。
- 【输出】与输入对应的一个四元式序列。
- 【题目】对实验六的程序进行升级改造，使得程序对于输入的一个完整的源程序，在对其做递归下降分析的同时，生成等价的四元式序列，一遍完成。

最后一周查缺补漏

附录A

基于C语言设计的源语言LittleC

该语言的一个程序由一个块组成，该块中包含可选的声明语句和执行语句。该语言只支持一种数据类型：整型，因而也只有一种类型的变量：整型变量和一种类型的常量：整常数。该语言支持 +、-、*、/四种算术运算，运算式的值为整常数。该语言的选择语句和循环语句中允许使用关系运算表达式来作为控制条件，其值为整常数（非零值表示true，零值表示false）。运算符的优先级和结合规则按照C语言语法理解。该语言不支持数组、结构体、指针等复杂数据类型，不支持函数调用，没有I/O功能。

1. 文法定义

```
PROG→BLOCK
BLOCK→{  DECLS  STMTS  }

DECLS→DECLS  DECL | empty
DECL→TYPE  NAMES  ;
TYPE→int
NAMES→NAMES  ,  NAME | NAME
NAME→id

STMTS→STMTS  STMT | empty
STMT→id  =  EXPR ;
STMT→if     ( BOOL )      STMT
STMT→if  (  BOOL )  STMT  else  STMT
STMT→while  (  BOOL )  STMT
STMT→BLOCK

EXPR→ TERM EXPR1
EXPR1→ ADD TERM EXPR1 | empty
TERM→ FACTOR TERM1
TERM1→ MUL FACTOR TERM1 | empty
FACTOR→ id | number | (EXPR)
ADD→ + | -
MUL→ *| /
BOOL→ EXPR ROP EXPR
ROP→ > |>= | < | <= | ==| !=
```

2 词法规则

1. ）名字：由字母打头后跟字母、数字任意组合的字符串；长度不超过20；不区分大小写；把下划线看作第27个字母；
2. ）常数：完全由数字组成的字符串；正数和0前面不加符号，负数在正数前面加-构成；长度不超过15；
3. ）关键字、运算符、分隔符仅包含在文法定义中出现过的单词。
4. ）字母表定义为 1.）～ 3.）中出现的字符的集合；不在该集合中的符号都以非法字符对待；

基于Pascal语法设计的源语言LittleP

由于Pascal语言结构严谨，层次清晰，语法与C语言接近，也便于理解，因此本实验抽取Pascal语言的一个子集，稍加改造，作为源语言，姑且命名为LittleP。一个LittleP程序由一系列全局数据声明和一个主程序体组成。所有数据采用静态存储分配，没有I/O，只支持一种基本数据类型：无符号整数。该语言支持 +、-、*、/四种算术运算，运算式的值为整常数。该语言的选择语句和循环语句中允许使用关系运算表达式来作为控制条件，其值为整常数（非零值表示true，零值表示false）。运算符的优先级和结合规则按照C语言语法理解。

该语言不支持数组、结构体、指针等复杂数据类型，不支持函数调用。

1. 文法定义

```
<程序>→<程序首部><程序体>.
<程序首部>→ program<程序名>;
<程序体>→<变量声明><复合语句>

<变量声明>→ var<变量定义列表>|<空>
<变量定义列表> → <变量定义列表> ; <变量定义>|<变量定义>
<变量定义>→<变量名列表>: <类型>
```

```
<变量名列表> → <变量名列表>,<变量名>|<变量名>
<类型> →  integer

<复合语句>→ begin <语句块> end
<语句块>→<语句>丨<语句块> ； <语句>
<语句>→<赋值语句>|<条件语句>|<循环语句>|<复合语句>|<空>

<赋值语句>→<左部>:= <右部>
<左部>→<变量名>
<右部>→<算术表达式>
<条件语句>→ if（<关系表达式>）then<语句>else<语句>
<循环语句>→ while（<关系表达式>）do<语句>

<关系表达式> →<算术表达式><关系运算符><算术表达式>
<算术表达式> → <项>| <算术表达式><加运算符><项>
<项> → <因子>| <项><乘运算符><因子>
<因子>→<变量名>丨(<算术表达式>) 丨<整数>

<程序名>→<标识符>
<变量名>→<标识符>
<标识符>→<字母>丨<标识符><字母>丨<标识符><数字>

<整数>→<数字>丨<整数><数字>
<关系运算符>→ < | <=  | > | >= | ＝ | <>
<加运算符>→ + | -
<乘运算符>→ * | /
<字母>→ a|b|…|x|y|z
<数字>→ 1|2|3|4|5|6|7|8|9|0
```

2. 词法规则
    1. ）名字：由字母打头后跟字母、数字任意组合的字符串；长度不超过15；不区分大小写；
    2. ）常数：完全由数字组成的字符串；没有正负号；长度不超过15；
    3. ）关键字、运算符、分隔符仅包含在文法定义中出现过的单词。
    4. ）母表定义为（1）∽（3）中出现的字符的集合；不在该集合中的符号都以非法字符对待；

附录B

对自定义源语言的要求：

1. 该语言应该至少包含一种数据类型：整型，能够表达整型的变量和常量；
2. 能够进行简单的算术运算；
3. 能够表达三种常用的语义结构：顺序、选择和循环；
4. 程序有唯一的入口/起点和唯一的出口/终点；