華南師範大學

本科学生实验（实践）报告

院　　系：计 算 机 学 院

实验课程：编译原理项目

实验项目：编译原理课程夏目

指导老师：黄煜廉

开课时间：2022 ～ 2023 年度第 2 学期

专　　业：网络工程 2 班

班　　级：2020 级

学　　生：陈建宇

学　　号：20202132045

华南师范大学教务处

华南师范大学实验报告

学生姓名：陈建宇　　学　　号：20202132045
专　　业：网络工程　　年级、班级：2020 级网工 2 班
课程名称：编译原理项目　　实验　项目：编译原理课程项目
实验类型：口验证　√设计　口综合　　实验　时间：2023 年 6 月 20 日
指导老师：黄煜廉　　实验　评分：

一、项目内容

任务要求一

（1）以文本文件的方式输入某一高级程序设计语言的所有单词对应的正则表达式，系统需要提供一个操作界面，让用户打开某一语言的所有单词对应正则表达式文本文件，该文本文件的具体格式可根据自己实际的需要进行定义。

（2）需要提供窗口以便用户可以查看转换得到的 NFA（可用状态转换表呈现）

（3）需要提供窗口以便用户可以查看转换得到的 DFA（可用状态转换表呈现）

（4）需要提供窗口以便用户可以查看转换得到的最小化 DFA（可用状态转换表呈现）

（5）需要提供窗口以便用户可以查看转换得到的词法分析程序（该分析程序需要用 C 语言描述）

（6）对要求(5)得到的源程序进行编译生成一个可执行程序，并以该高级程序设计语言的一个源程序进行测试，输出该该源程序的单词编码。需要提供窗口以便用户可以查看该单词编码。

（7）对系统进行测试:

（A）先以 TINY 语言的所有单词的正则表达式作为文本来测试，生成一个 TINY 语言的词法分析源程序；

（B）接着对这个词法分析源程序利用 C/C++编译器进行编译，并生成可执行程序；

（C）以 sample.tny 来测试，输出该 TINY 语言源程序的单词编码文件 sample.lex。

（8）要求应用程序为 Windows 界面

（9）书写完善的软件文档

任务要求二

（1）以文本文件的方式输入某一高级程序设计语言的所有语法对应的 BNF 文法，因此系统需要提供一个操作界面，让用户打开某一语言的所有语法对应的 BNF 文法的文本文件，该文本文件的具体格式可根据自己实际的需要进行定义。

（2）需要提供窗口以便用户可以查看文法化简后的结果（可用表格形式进行呈现）

（3）需要提供窗口以便用户可以查看消除左公共因子和左递归之后的新文法（可用表格形式进行呈现）

（4）求出改造后文法的每个非终结符号的 First 集合和 Follow 集合，并需要提供窗口以便用户可以查看该结果（可用两张表格的形式分别进行呈现）

（5）构造出 LL(1)分析表，并需要提供窗口以便用户可以查看该结果（可用表格形式进行呈现）

（6）采用 LL(1)语法分析方法进行语法分析并生成相应的语法树，每个语句的语

法树结构可根据实际的需要进行定义。 （语法树需要采用树状形式进行呈现）

（7）对系统进行测试：以 TINY 语言的所有语法以及第一项任务的测试结果 sample.lex 作为测试，进行 LL(1)语法分析并生成对应的语法树。

（8）要求应用程序为 Windows 界面

（9）书写完善的软件文档

任务要求三

mini-c 语言作为测试

（1）以 mini-c 的词法进行测试，并以至少一个 mini-c 源程序进行词法分析的测试（该

mini-c 源程序需要自己根据 mini-c 词法和语法编写出来，类似于 sample.tny）。

（2）以 mini-c 的语法进行测试，并以测试步骤（1）的源程序所生成的单词编码文件进

行语法分析，生成对应的语法树。

选做内容

1.采用 LL(1)语法分析方法进行语法分析并生成相应的中间代码，中间代码可以自选（可以是四元组、三元组、伪代码等中间代码），中间代码生产结果可以在屏幕上显示或以文件的形式保存 。

编程要求

1.编程所使用的编程语言以及平台不做要求。

2.程序应该具有良好的编程风格

3.必要的注释；（简单要求如下）

1） readme 文件对上交的实验内容文件或目录作适当的解释；

2） 每个源程序文件中注释信息至少包含以下内容：

（1）版权信息。

（2）文件名称，标识符，摘要或模块功能说明。

（3）当前版本号，作者/修改者，完成日期。

（4）版本历史信息。 // （1）--（4）部分写在文件头

（5）所有的宏定义，非局部变量都要加注释

（6）所有函数前有函数功能说明，输入输出接口信息，以及调用注意事项

（7）函数关键地方加注释

二、项目目的

（1）理解有穷自动机及其应用。

（2）掌握 NFA 到 DFA 的等价变换方法、DFA 最小化的方法。

（3）掌握设计、编码、调试词法分析程序的技术和方法。

（4）设计、编制、调试、实现一个某一高级程序设计语言的语法分析程序，加深对语法分析原理的理解。

（5 通过设计调试语法分析程序，实现从源程序中分离出各种类型的单词；加深对课堂教学

的理解；提高语法分析方法的实践能力。

（6）掌握从源程序文件中读取有效字符的方法和产生源程序的内部表示文件的方法。

三、项目文档：

引言

背景

编译器是将一种程序语言（源程序：source language）翻译为另一种程序语言（目标程序：target language）的计算机程序。一般来说，源程序为高级语言，而目标语言则是汇编语言或机器码。

早期的计算机程序员们用机器码写程序，编程十分耗时耗力，也非常容易出错，很快程序员们发明了汇编语言，提高了编程的速度和准确度，但编写起来还是不容易，且程序严格依赖于特定的机器，很难移植和重复利用。

上世纪 50～60 年代，第一批高级语言及编译器被开发出来，程序的文法非常接近于数学公式以及自然语言，使得编写、阅读和维护程序的难度大为降低，程序编制的速度、规模和稳定性都有了质的飞跃。

可以说是编译器的产生带来了计算机行业的飞跃发展，所以开发编译器是非常有必要的。

整体设计

运行环境

语言：C++ 14

集成开发环境：Visual Studio 2022、Qt Creator 8.0.1

界面库：Qt 5.15.2

编译环境：MSVC 2019 x64

开发平台：Windows

编译方法

正则表达式规则

- 支持单个字符
- 连接 &
- 选择——|
- 闭包——*
- 正闭包——+
- 可选——?
- 括号——()

文法文件规定

- 文件类型：文本文件(.txt)
- 文件内容：文法规则
- 内容格式：一行只存储一条规则，不得有非必要空格符，区分大小写
- 指定符号：为方便处理，单一个大写字母作为非终结符号，单一个小写字母作为终结符号，用@表示空串。

输入模块

功能

程序输入输入均是文件形式。

1. 选择待分析的编程语言类型（目前只支持 Tiny、Mini-C）
2. 选择输入的正则表达式文件并显示（支持对话框选取或拖放）
3. 选择输入的 BNF 文法文件并显示（支持对话框选取或拖放）
4. 选择输入的源代码文件并显示（支持对话框选取或拖放）

数据结构

- QFile：读取文件信息（绝对路径等）
- QTextStream：读取文件内容（readAll()）
- QLineEdit:显示文件绝对路径
- QTextBrowser:显示文件内容

流程逻辑

1. 由用户选择必要的输入信息（语言类型、正则表达式、BNF 文法、源程序）
2. 用户可点击按钮，通过文件选择对话框选取
3. 用户可拖拽文件至对应的显示框内

输出模块

生成文件：

- 词法分析程序源码：tiny-lex.c, minic-lex.c
- 词法分析可执行程序：tiny-lex.exe, minic-lex.exe
- 单词编码文件：tiny-token.txt, minic-token
- 语法树文件：tiny-tree.txt, minic-tree.txt

存储位置

- 执行已打包程序：内嵌
- IDE 中运行："项目可执行程序目录"/save_tiny/、"项目可执行程序目录"/save_minic/

词法分析模块

功能

1. 将某高级程序设计语言所有单词的正则表达式->NFA
2. NFA->DFA 最小化 DFA
3. 最小化 DFA->词法分析程序
4. 该语言的一个源程序->单词编码

输入输出项

- 输入：存有某高级程序设计语言所有单词的正则表达式的文本文件、该语言源程序
- 输出：NFA、DFA、最小化 DFA（表格形式），词法分析程序源程序（C 语言描述），词法分析可执行程序，单词编码（文本文件）

数据结构

- vector<string>：正则表达式
- vector<string>：表达式前缀

- map<int, map<string, set<int> > >：NFA
- map<string, map<int, map< string, set<int> > > >：表达式对应的 NFA
- map<string, pair<int, int> >：所有表达式对应的 NFA 的头尾节点

- map<set<int>, map<string, set<int> > >：DFA
- map<string, map<set<int>, map<string, set<int> > > >：表达式对应的 DFA
- map<int, map<string, int> >：名称简化后 DFA 对应关系

- map<int, map<string, int> >：最小化的 dfa 对应关系
- map<string, map<int, map<string, int> > >：最小化 dfa 关系组

算法

正则表达式—>NFA

正则表达式递归定义：

给定字母表 Σ，Σ 上的正则表达式由且仅由以下规则定义：

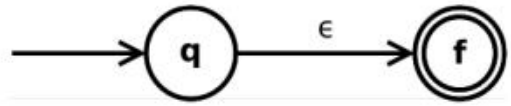
1. ϵ 是正则表达式；
2. ∀a ∈ Σ，a 是正则表达式；
3. 如果 r 是正则表达式，则 (r) 是正则表达式；
4. 如果 r 与 s 是正则表达式，则 r|s，rs，r* 也是正则表达式。

Thompsion 构造法

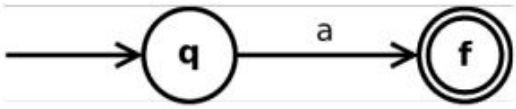
由正则表达式通过 Thompson 构造 NFA 有三种情况

1. 状态机一定只有一个初始状态节点和一个结束状态节点。
2. 任何一个状态，最多只有两条出去的转换边。
3. 每个状态节点所拥有的边最多只有三种可能：

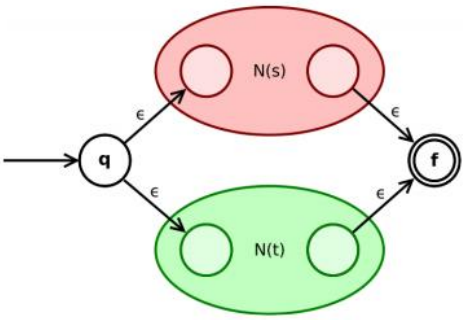
- 基本规则：表达式 ϵ



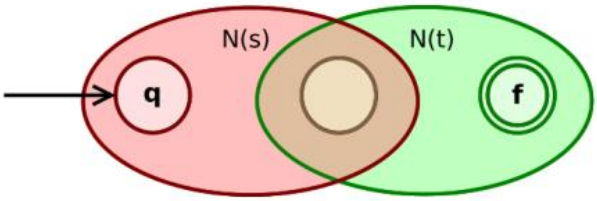
- 基本规则：表达式 a ∈ Σ



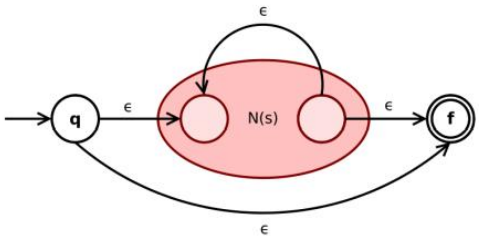
- 归纳规则：表达式 r = s|t



- 归纳规则：表达式 r = st



- 归纳规则：表达式 r = **r***



-

NFA—>DFA

子集构造法

从状态 s 开始，只通过 ϵ-转移可达的状态集合

$$\epsilon\text{-closure}(s) = \{t \in S_N | s \xrightarrow{\epsilon^*} t\}$$

$$\epsilon\text{-closure}(T) = \bigcup_{s \in T} \epsilon\text{-closure}(s)$$

$$\text{move}(T, a) = \bigcup_{s \in T} \delta(s, a)$$

求 ε 闭包（ ε -closure），通过图的深度遍历方法。伪代码描述如下

```
将 T 的所有状态压入 stack 中;
将 ε -closure(T)初始化为 T;
while(stack 非空) {
    将栈顶元素 t 给弹出栈中;
    for(每个满足如下条件的 u:从 t 出发有一个标号为 ε 的转换到达状态 u)
        if (u 不在 ε -closure(T)中) {
            将 u 加入到 ε -closure(T)中;
            将 u 压入栈中;
        }
}
```

从开始节点的 ε 闭包开始，对每一个转换字符求取转换结果，将新的转换节点加入新增节点集合 Q 中，并对新增节点重复上述操作。直到不再产生新的节点，则新增节点集合 Q 构造完毕，Q 即为 DFA 的节点集合。

以上是子集法的大致描述，为了实现上面的算法，我们需要一些数据结构来存储相应的内容：

1. 新增节点集合 Q：vector<set<char>> Q
2. 不再产生新节点:queue<set<char>> s

注意：

NFA 转 DFA 的时候，经常需要在 DFA 集合中加入一个死状态。所谓死状态就是任何输入下都指向自己的状态。如果进入死状态就代表输入字符串已不符合该正则表达式的规则。

DFA 最小化

核心思想：如果状态等价，则将其合并。

采用 Hopcroft 算法，即将集合不断划分，直至不再产生新集合为止，伪代码描述如下：

```
split(S)
    foreach(character c)
        if(c can split s)
            split s into T1, ..., Tk

hopcroft()
    split all nodes into N, A
    while(set is still changing)
        split(s)
```

核心问题：如何定义等价状态？

重点在于集合的划分。

将一个 DFA 的状态集合划分为多个组，每个组中的各个状态之间相互不可区分。然后，将每个组的状态中的状态合并成状态最少 DFA 的一个状态。算法在执行过程中维护了状态集合的一个划分，划分中的每个组内的各个状态尚不能区分，但是来自不同组的任意两个状态是可区分的。当任意一个组都不能再被分解为更小的组时，这个划分就不能再进一步精化，此时我们就得到了状态最少的 DFA。

最初，该划分包含两个组：接收状态组和非接受状态组。算法的基本步骤是从当前划分中取一个状态组，比如 A={s1, s2, …, sk}，并选定某个输入符号 a，检查 a 是否可以用于区分 A 中的某些状态。我们检查 s1, s2, …, sk 在 a 上的转换，如果这些转换到达的状态落入当前划分的两个或多个组中，我们就将 A 分割成为多个组，使得 si 和 sj 在同一组中当且仅当它们在 a 上的转换都到达同一个组的状态。我们重复这个分割过程，直到无法根据某个输入符号对任意个组进行分割位置。
伪代码描述算法如下：

```
// 最初，构造出示划分
for(划分中的每个组 G) {
    将 G 划分为更小的组，使状态 s 和 t 在同一小组中，当且仅当对于所有的输入符号 a，状态 s 和 t 在 a 上的转换都达到当前划分中的同一组;
    // 最坏情况下，每个状态各自成一个组
    更新当前划分;
}
```

重复上述代码，直到没有新的划分产生。然后每个组都选取一个代表。最小化 DFA 的开始状态是包含 D 的开始状态的组的代表。

由于上述的分组在每次拆分后会从头进行检查，并且不时的要删除与添加新的集合，所以这里不能像 ε -closure 子集法那里一样，在求新的节点时顺便将边集保存，我们需要根据新的节点求新的边集。

需要求取新的节点在原本的 DFA 中的转换结果集合，然后检查该结果属于哪一个新的节点的子集，从而获得新的边集，伪代码描述算法如下：

```
for s,i in all_set{
    for ch in transchar{
        set<char> trans_result = transfer(s,ch);
        // 如果转换结果不为空，加入边集
        if(trans_result.size()!=0)
        {
            // 找出转换结果属于哪一个集合的子集
            j = sub_set(all_set,trans_result);
            dfa.transet[i][j] += ch;
        }
    }
}
```

最小化 DFA—>词法分析程序

- 从最小化 DFA 图的起点开始遍历即可，DFA 的初态（即起点）一定是唯一的，而接收态（即终点结点）可能有多个（考虑正则表达式”a|b”）。
- 进行遍历，如果边指向自己则是 while 语句，先翻译 while 语句，然后对于每条前进边都是 if-else 语句
- 此函数通过一步一步分析将生成 C 语言语句的每一行用一个字符串存储，并将这些字符串存储在 vector 类型的 lines 中，方便后续处理，将这些语句写到 txt 文件。

```
void nfaManager::getCcode(int v, vector<string> &lines)
{
    if(mini_dfa.NodeTable[v].final==1)
        lines.push_back("Done();");

    //参数 v 为现在处理的结点在 mini_dfa 图中的序号
    //首先收集结点 v 有多少条指向自己的边，收集该边的 dest
    vector<char> while_char,if_char;
    Edge *p=mini_dfa.NodeTable[v].adj;
    while(p!=NULL)
```

```
    {
        if(p->nextVertex==v)
            while_char.push_back(p->dest);
        else
            if_char.push_back(p->dest);
        p=p->link;
    }

    if(!while_char.empty()) //如果不为空，说明存在指向自身的边，这个时候就要生成while语句
    {
        //生成的对应C语言分析程序，首句为“char ch=getChar()”
        lines.push_back("char ch = getChar();");
        string line="while(";
        int i=0;
        for(;i<while_char.size()-1;)
        {
            line +="ch ==";
            string str;
            str+=while_char[i];
            line+=str+"||";
            i++;
        }
        string str;
        str+=while_char[i];
        line+="ch =="+str+")";
        lines.push_back(line);
        lines.push_back("{");
        if(mini_dfa.NodeTable[v].final==1)
            lines.push_back("Done();");
        lines.push_back("ch = getChar();");
        lines.push_back("}");
        if(if_char.empty())
        {
            lines.push_back("error;");
        }
    }

    //处理完指向自身结点的边后，接下里处理指向别的结点的边
    if(!(if_char.empty()))
    {
    if(while_char.empty())
        lines.push_back("char ch = getChar();");
    Edge *q= mini_dfa.NodeTable[v].adj;
```

```
    if(q!=NULL)
    {
        while(q!=NULL)
        {
            if(q->nextVertex!=v)
            {
                string line_1="if( ch ==";
                string str_2;
                str_2+=q->dest;
                line_1+=str_2+")";
                lines.push_back(line_1);
                lines.push_back("{");
                getCcode(q->nextVertex,lines);
                lines.push_back("}");
                lines.push_back("else");
            }
            q=q->link;
        }
        string str_3;
        str_3+=48+v;
        string line_2;
        line_2+="error("+str_3+");";
        lines.push_back(line_2);
    }
    }
}
```

运行程序进行语法分析

使用 MinGW64gcc 编译器进行编译，生成可执行程序。

通过 QProcess 执行词法分析程序

语法分析模块

功能

输入输出项

- 输入：存有某高级程序设计语言 BNF 文法的文本文件、该语言源程序
- 输出：

数据结构

- string：符号
- vector<string>：终结符号集合
- vector<string>：非终结符号集合
- map<string, vector<string> >：产生式
- map<string, set<string> >：First 集合
- map<string, set<string> >：Follow 集合
- map<pair<string, string>, string>：LL(1)文法分析表

算法

BNF 文法化简

文法化简三个过程：

（1）无用符号和无用产生式的删除

基本算法：对于任意上下文无关文法 G=(V,T,P,S),w∈L(G),X∈V,若存在 a,b∈(VUT)*使得 S 经过若干步推出 aXb，aXb 经过若干步推出 w,则称 X 为有用符号，否则为无用符号。

1、计算“产生的”符号集 N：每个 T 中的符号都是产生的，若 A→a∈P 且 a 中符号都是产生的，则 A 是产生的。
伪代码:

```
for (int i=0;i<V.num;i++)
     for(int j=0;j<v.num;j++)//v 表示 V 的拷贝
        if(p 中存在 v[j] →a&&a 中的每个符号都属于 N)
          {
            将 v[j]从 v 中删除并加到 N 中，同时跳出内层循环}
```

2、计算“可达的”符号集 M：开始符号 S 是可达的，A→a∈P 且 A 是可达的，则 a 中的符号都是可达的。
伪代码：

```
Reach(S)
{
    if(P 中存在 S→a) {
        将 a 中的所有字符加入到 M 中；
            if(a 中存在非终结符 B)
                  Reach(B)
            }
    return
}
```

3、消除全部非“产生的”符号，在消除全部非“可达的”符号，剩下的都是有用符号。最后将无用字符和无用产生式都删除。

伪代码:

```
for(int i=0;i<Q.num;i++){//Q 是 VUT 的集合
     for(int j=0;j<N.num;j++){
         if(Q[i]不在 N 中)
            将其加入到非“产生的”符号集 N1 中}}
//  消除全部非“可达的”符号
for(int i=0;i<N1.num;i++){
   for(int j=0;j<M.num;j++){
      if(N1[i]不在 M 中)
         将其加入到无用符号集 NM 中}}
//消除产生式和无用符号，因结构类似只写了一个
for(int i=0;i<P.num;i++){
    if(P[i]个产生式中含有 NM 符号集的元素)
        将该条产生式删除}}
```

（2）epsilon 产生式(空产生式)的删除

定义：称形如 A→ ε 的产生式为 ε 产生式。

1、由文法推出满足定义（A∈V,且 A 能在有限步推出 ε ）的非终结符集合 V1 伪代码

```
for(int i=0;i<V.num;i++)
    for(int j=0;j<V.num;j++){
        if(左部为 V[i]的产生式右部所有符号都在 V1 中)
            将 V[i]加入 V1 中，跳出内层循环；
```

2、若产生式 B→aOB1…Bkak∈P,其中 aj∈(VUT)*,Bi∈V1,那么用 ε 和 Bi 本身代替 Bi。然后将这些产生式都加入到新的产生式集合中，不满足上述产生式的直接将空产生式扣除后加入

到新产生式集合中。若有 S→ ε ，则引入 S|→ ε |S。S|为新的开始符号。

```
for(int i=0;i<P.num;i++)
    if(存在产生式 B→a0B1...Bkak∈P，aj∈(VUT)*,Bi∈V1)
        以 Bi 或者空代替 Bi，并将形成的产生式加入到 P|中;
    else if(产生式左部属于 V1)
        将 ε 产生式去除后将其加入 P|中;
    else if(若有 S→ ε )
        引入 S|→ ε |S。S|为新的开始符号;}
```

（3）单一产生式的删除

思路：如果存在产生式 A→B，则将 B 作为 A 的子节点加入图集合 X 中，若存在 B→C,则同样将 C 作为 B 的子节点加入到 X 中。所有的非终结符都进行这样的处理，然后对图集合 X 进行遍历，所有子节点加入到祖宗节点的链集合中，比如 A 的子节点有 B、C、D，则将 B、C、D 加入 A 的链集合中。对于每个非终结符的链集合中若存在非终结符（假设为 B），且 B→w 属于 P ,w 不属于 V，则将 A→w 加入到 P|中。遍历所有的链集合后便完成了单一产生式的消除。

伪代码描述如下：

```
for(int i=0;i<P.num;i++)
  if(存在单一产生式（A→B）)
将 B 作为 A 的子节点加入到图中;
遍历图找到每个非终结符的链集合;
遍历每个非终结符的链集合：
如果非终结符的链集合不为空（例如 B 在 A 的链集合中）：
    且 B→w 属于 P,w 不属于 V;
     将 A→w 加入到 P|中;
如果非终结符的链集合为空则不进行处理。
```

消除左递归

1、消除直接左递归：A→Aa

对于 A→Aa|b（b 可为空）。因为推导结束后一定有个 b 在开始位置，故改为：A→bB,B→aB。

2、消除间接左递归：P→Aa,A→Pb

0）对所有非终结符（Non-Terminator）随机编号排序
1）若消除过程中出现直接左递归，则依据直接左递归的方法消除
2）若产生式右部最左的符号是 Non-Terminator，且该非终结符的序号>=左部非终结符，则暂不处理（后续会处理）

3）若序号<左部的非终结符，则用之前求得的式子的右部来替换

所有非终结符按任意顺序排列编号[V1,…,Vn]
按上面的排列顺序，对这些非终结符遍历

```
for(int i = 1; i <= n; ++i) {
  for(int j = 1; j <= n - 1; ++j) {
     遍历产生式，若 V[j]序号小于等于产生式右部非终结符按规则 3）
     进行替换（序号大于的按规则 2）处理） }
  消除 i 序号的非终结符的直接左递归（如果存在的话）
}
```

这里需要注意的是，消除左递归后会产生空产生式和无用符号，
所以消除左递归后需要在将空产生式和无用符号处理一遍。

提取左公因子

思路：有相同前缀的多个产生式，除去相同前缀，剩余部分联合起来组成一个新的产生式组，让它成为一个新的非终结符对应的候选式组。

伪代码：

```
while(每个非终结符任意两个产生式体有公共前缀)
{
对于每个非终结符号 A，找出其多个候选式的最长公共前缀α；
    if(α ≠ ε) { // 存在一个非平凡的(nontrivial)公共前缀
        将所有 A-产生式（A → α β1 | α β2 | ... | α βn |γ1|γ2|...|γm）
        替换为
        A → αA′ |γ1|γ2|...|γm
        A′ → β1|β2|...|βn
// 其中，γi 表示：所有不以α开头的产生式体；
        A′ 是一个新的非终结符
    }
}
```

求 Fisrt、Follow 集

求 First 集算法流程

1.若 X->a…，则将终结符 a 放入 First（X）中
2.若 X->ε，则将 ε 放入 First（X）中
3.若有 X->Y1Y2Y3…Yk，则
（1）把 First（Y1）去掉 ε 后加入 First（X）
（2）如果 First（Y1）包含 ε，则把 First（Y2）也加入到 First（X）中，以此类推，直到某一个非终结符 Yn 的 First 集中不包含 ε

（3）如果 First（Yk）中包含 ε，即 Y1~Yn 都有 ε 产生式，就把 ε 加入到 First（X）中。

算法描述：

- 遍历每一个左部为 x 的产生式
- 如果产生式右部第一个字符为终结符，则将其计入左部非终结符的 First 集
- 如果产生式右部第一个字符为非终结符
- 求该非终结符的 First 集
- 将该非终结符的去掉**$**的 First 集计入左部的 First 集
- 若存在**$**，继续往后遍历右部
- 若不存在**$**，则停止遍历该产生式，进入下一个产生式
- 若已经到达产生式的最右部的非终结符（即右部的 First 集都含有空串），则将**$**加入左部的 First 集
- 处理数组中重复的 First 集中的终结符

求 Follow 集算法流程

1.对于文法开始符号 S，把$加入到 Follow（S）中
2.若有 A->aBC，则将 First（C）中除了 ε 之外的元素加入到 Follow（B）（此处 a 可以为空）
3.若有 A->aB 或者 A->aBC 且 ε 属于 first（C），则将 Follow(A)加入到 follow（B）中（此处 a 可以为空）
4.若有 A->Bc,则直接将 c 加入 follow（B）中 follow 集中不含有空串 ε

- 遍历每一个右部包含非终结符 x 的产生式
- 如果 x 的下一个字符是终结符，添加进 x 的 Follow 集
- 如果 x 的下一个字符是非终结符,把该字符的 First 集加入 x 的 Follow 集(不能加入空串)
- 如果下一字符的 First 集有空串并且该产生式的左部不是 x，则把左部的 Follow 集加入 x 的 Follow 集
- 如果 x 已经是产生式的末尾，则把左部的 Follow 集添加到 x 的 Follow 集里

求 LL(1)分析表

构造 LL(1)分析表算法思想

- 遍历每一个产生式
- 如果右部的第一个字符 tmp 是终结符且不是空串，更新预测分析表，即 table[left] = i（i 为产生式编号）
- 如果右部的第一个字符是空串，遍历左部的 Follow 集，更新预测分析表，即 table[left] = i（i 为产生式编号，x 为 Follow 集字符编号）
- 如果右部的第一个字符是非终结符，遍历它的 First 集，更新预测分析表，即 table[left] = i（i 为产生式编号，x 为 First 集字符编号）

1、对文法 G 的每个产生式 A → α 执行 2 和 3 步；
2、对每个终结符 a ∈ FIRST(A)，把 A → α 加至 M[A, a]中，其中 α 为含有首字符 a 的候选式或唯一的候选式
3、若 ε ∈ FIRST(A)，则对任何 b ∈ FOLLOW(A) 把 A → ε 加至 M[A, b]中
4、把所有无定义的 M[A，a]标“出错 标志”。

语法分析生成语法树

根据已有的 LL(1)分析表，利用栈进行求解。

输出：采用缩进的方式表示语法树的结构，每层缩进表示语法树的一层，一行表示一个节点，同一缩进下的节点表示位于语法树的同一层。

算法思路：

语法分析程序依据栈顶符号 X 和当前输入符号运行。对于任何(X,a),总控程序每次有以下三种可能得操作：

1、若 x=a=″#″，则分析成功，分析器停止工作

2、若 x=a≠″#″（即栈顶符号 x 与当前扫描的输入符号 a 匹配），则 x 出栈，让输入指针 a 指向下一个输入符号，继续对下个字符进行分析；

3、若 x 是一个非终结符 A，则查分析表 M[A,a]：

①若 M[A,a]中有关于 A 的一个产生式，则 A 出栈，并将 M[A,a]中产生式的右部符号串按逆序逐一入栈

②若 M[A,a]中产生式为 A→ ε，则将 A 出栈

③若 M[A,a]中产生式为空，则发现语法错误，调用出错处理程序进行处理。

四、实验总结（心得体会）

通过完成词法分析程序的过程,更加深入的了解和复习了 C++的使用方法,某些函数的运用,以及字符串的提取判别,文件流读取操作等等,也在调用函数过程中出现了许多的问题,通过请教同学和查阅资料都得到了解决.学习了新的函数,对 C++有了更加充实的认知.

深入理解了 LL1 文法。熟悉掌握了基于 LL1 文法的语义分析流程。针对特定文法，需要先计算出其 First 及 Follow 集合构建 LL1 文法分析表，并通过分析栈来实现一步步分析。

不过实际上很多核心算法都是先上网上临摹一遍别人的代码，自己再钻研一遍，才能学会，熟练度还有待提高，而且这一次的代码复杂程度远远超过我的想象，之后会更加努力地学习核心算法的实现和运行，同时也需要学习更多的 Qt 的使用方法。

改进方面和建议：

- 采用批量选择的方式输入文件
- 检查用户输入的文件类型，是否与当前要分析的源码类型匹配，或禁止用户输入不匹配的文件
- 优化数据结构，使得用户不重新打开软件，就能切换分析的语言
- 增加对于输入正则表达式和 BNF 文法的详细描述
- 采用图形化方法描述生成的 NFA、DFA、最小化 DFA 状态转换图和语法树

五、参考文献：

(29 条消息) 编译原理——词法分析器 C++实现 weixin_45693492 的博客-CSDN 博客

(29 条消息) （C++）带你手肝词法分析器，容易理解，跟着思路有手就行 *c++界符*Gassing 仔的博客-CSDN 博客

仿 lex 生成器（qt+C++实现） - 头发乱了 88 - 博客园 (cnblogs.com)

(29 条消息) 编译原理——词法分析器 C++实现 weixin_45693492 的博客-CSDN 博客

(29 条消息) 自己动手写编译器之 Tiny 语言语法分析器的实现 *sample 语言编译器* bigconvience 的博客-CSDN 博客

六、项目自评

项目完成情况：

在规定时间内上交实验程序及文档,完成了必做内容要求中的全部内容，文档规范，编程风格好，设计思想基本清晰，界面美观大方，使用方便。

自评分分数：80