大家好，我是czy，我小组的另外两名成员分别是xzh和cyt。接下来将先由我汇报一下我们小组项目的总体情况，之后会按照分工依次介绍工作细节，最后给大家展示我们生成的Lex+Yacc。

首先，项目总体情况。我们的工作主要分为两个阶段，即SeuLex和SeuYacc的设计。词法分析部分，分成五个模块，一：lex文件解析器，主要用于提取C语言标准下的正则表达式与词法规则，二：正则表达式规范化模块，我们将提取出的非规范正规表达式规范化，从而用于第三个模块：构建NFA部分，每条正则表达式转换为NFA，再将NFA合并，输入到第四个模块：构建DFA，主要包含NFA确定化与DFA的最小化两项工作，最后完成词法分析代码生成模块，得到我们生成的Lex文件。第二阶段的语法分析部分，更为简明，同理SeuLex第一个模块为Yacc文件解析器，提取出终结符、非终结符以及语法规则，用于第二个LR(1)项目集构造模块，基于此模块，LR(1)项目集规范族得以构建，二三模块交替进行，最终得到ACTION-GOTO分析表，辅助语法分析器代码生成模块，得到Yacc.cpp。

接下来，我概要介绍一下两阶段工作成果的整体功能。我们整个项目以99年推出的c99版本为标准，并非采用Cminus，以期望分析成功尽可能多类型的C语言代码。

由此，SeuLex的编译对象为老师提供的c99.l文件，为便于后续文件解析，我们在不改变词法的前提下稍作修改，主要体现在四个部分，一：.......一对百分号括号.......，这样规范文件格式后便于我们的SeuLex完成出错检查与处理；二：......如原文件中存在个别正则表达式的表示符号未用花括号声明，这样会导致它是表示符号还是本身符号的混淆歧义；三，......为方便后续词法分析代码生成不出错，能与Yacc连用，我们规定了动作代码中return后面一律返回字符串；四：......从而基于这样的Lex输入文件，我们SeuLex依次完成了之前所述五个模块的功能，这里不再重复阐述，而第一阶段工作的主要特色主要有关正则表达式，最长匹配原则是基本，依托于c99得以支持正则表达式四大元符号，在设计方面，我们运用C++的regex库匹配正则表达式，支持正则别名定义等等，下面罗列的细节可见我们小组的课程报告。

再来介绍SeuYacc的整体功能，同理我们选用了c99.y输入文件，同样需要对其在不改变语法的前提下稍作修改，才能实现对C99版本C语言进行语法分析。主要修改，第一仍是规范输入文件格式，便于后续出错处理。第二点比较特殊，原文件中以“%start”标记了文法开始符号是translation\_unit，而为了省去在语法规则部分检索文法开始符号的步骤，提高算法速度，我们与原版Yacc的行为保持一致，如果定义段中没有“%start”的说明，Yacc自动将语法规则部分中第一条语法规则左部的非终结符作为语法开始符，从而我们修改为将translation\_unit开头的语法规则移到文件中语法规则部分第一条即可。第三点，......，这样才能解决语法分析过程存在的可能冲突与二义性。修改完毕后，我们的SeuYacc得以完成了老师要求的语法分析基本要求部分，接着再进行拓展工作中的主要特色，除了我们考虑到用LALR(1)分析，遇到输入有错时并不能及时报错，而是会多做了几步归约后才发现，而用LR(1)分析，能够遇到错误立即报错，我们便没有将LR(1)分析简化为LALR(1)分析，而完成了下面的四点拓展，相比于Cminus我们解析的c99本身语言子集足够强大，同时符号表的构建与相应管理程序以及出错处理，我们在生成语法分析代码模块时认为都是必须的，最后对于语义动作程序，我们的SeuYacc的输入文件解析模块会查找是否存在动作代码部分，并且允许其使用dollar（$）来获取栈内元素。

以上便是我们小组项目的一个总体情况，接下来按照分工我们组三个人会轮流介绍SeuLex与SeuYacc的设计细节，就实验内容、实验中主要数据结构、主要算法与遇到的问题及解决方案展开。

先是SeuLex工作。

我负责的DFA模块，主要实验内容：接受刚刚合并完成的一个整的NFA，进行NFA确定化与DFA最小化工作。

主要数据结构有三个，结构体Node第三个模块有讲过，DFA节点主要用了其中NFANodeSet集合，存的是NFA节点编号，从而完成一个DFA状态与NFA状态子集的对应。接着定义了一个DFA开始节点以及一个map结构的Symbols用于NFA确定化跳转表中每一行的构建。

主要算法即两部分，一：NFA2DFA()函数实现NFA确定化。

取第一个集合firstGroup只含NFA开始节点id，再利用findClosure函数构建这一集合的闭包，得以构建DFA的开始节点，加入DFA状态集Q；

接着，利用makeDFATable()函数构建NFA确定化过程中的跳转表，对于每个可能的输入符号a，我们能够得到一个转换后的状态集合I，再次利用findClosure函数构建这一集合的闭包，得到一个DFA节点，若状态集Q中不存在则要将其加入；

如此重复这第二步直到Q中不再有新的DFA节点出现为止，完成NFA的确定化过程。

二：minDFA()函数，对于刚刚确定化的DFA进行最小化工作。

我们选用了划分法，这个算法的主要思路为：......，

对于划分，我们首先以终态与非终态来划分，然后每次划分后对于每个子集考察是否还可以划分，若存在对于某一输入符号，得到的转换后状态集合不全包含于该子集中，该子集便仍存在不等价的状态，要一分为二；

重复如上操作，直到每个子集都是不可再分的了，我们从每个子集中任选一状态作为代表，消去其他等价状态，将那些原来射入其他等价状态的弧改射入相应的代表状态。子集中包含原有的初态时，其代表状态便为最小化后DFA的初态，包含原有终态时，代表状态便为最小化后DFA的终态。

由此完成了我们整个DFA模块。实验中主要有两个注意点：一来存在某个DFA 接收态包括多种动作代码不同的NFA接收态，......，二来......，解决了这两个问题，最小化的有限状态自动机便完成了。

下面是词法分析代码生成模块。

第二阶段的SeuYacc工作，首先是对c99.y文件解析模块。在这一部分就几乎定义全了整个SeuYacc全局需要的数据结构，灵活使用set，map，vector等数据结构，存储语法分析过程中需要用到的符号、产生式等，以及他们的优先级与左右结合关系。

解析文件的主要思路与输入文件的格式息息相关，c99.y主要分为四部分，第①和第④部分包含的头文件、宏定义、数据类型定义、全局变量定义以及用户子程序，均直接读入Yacc.cpp中即可，对于以百分号%声明的第②部分，我们会先读到第一个空格判断此定义是有关哪一内容，即是终结符token还是语义值类型type，或者是运算符的左右结合性声明，再逐行读取剩下内容，先判断是否存在用尖括号标记的语义值类型声明，之后逐空格截取字符读取每个终结符/非终结符的定义，给其编号并用map存储符号与编号一一对应。需要注意的是，这里......下面处理第③部分的语法规则，其主要构成为产生式+语义动作，......。

如此基于上述模块构建的LR(1)分析表，......

由此，我们便完成了整个自动生成Lex+Yacc的工作，接下来就展示一下我们的成果。......