

软件课程设计

Compiler-词法语法分析器

Fundamentals of Compiling

姓名： 李广通

学号： 919106840421

班级： 9191062301

专业： 计算机科学与技术专业

学院： 计算机科学与工程学院



南京理工大学
NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

目录

1. 概述

2. 开发工具

3. 开发设计思路

3.1 文法设计

3.2 编程实现

3.2.1 文件结构

3.2.2 数据结构、主要函数说明

4. 实例展示和效果说明

4.1 词法分析

4.2 语法分析

5. 心得体会

1.概述

课程任务目标：

任务 1：创建一个词法分析程序，该程序支持分析常规单词。

任务 2：创建一个使用 LL(1) 方法或 **LR(1) 方法**的语法分析程序。

任务 3（选作）：创建符合属性文法规则的语义分析程序。

完成情况：

根据老师的课设完成要求，前两项任务**全部个人独立开发设计完成（支持查重）**，并全部按起评分最高的推荐方法完成程序编写和文法设计，其中：

1.任务 1 词法分析程序在识别 5 种 token 的既定任务之外，为了最大限度地贴合真实高级语言要求规则，精心设计词法分析 2 型文法，**实现了对 token 中不同常量（整型、浮点型、复数）、关键字、界符、标识符的合法性检查以及分类**，并将常量细化为**科学计数法、10 进制、8 进制、16 进制、复数、普通浮点数、且全部支持各自的合法性检查**，通过文法设计和编程结构的优化**最大限度地贴合实际编程规范**。

2.任务 2 文法分析选择了起评分高的 **LR (1)** 分析法，实现基础要求的基础上，还实现了**语法错误的识别标注相关行号并输出错误信息**，结合任务一显示可能的错误信息，通过语法分析的文法设计，尽可能地**包含大部分机动性编程情况**，满足可以随意多类型变量声明、赋值、循环嵌套、多头文件声明、主函数单一出口、变参个数计算等功能，对任务 1 的输入要求更加的包容，过程信息运行时**即时输出各种信息**。

整个开发过程中的**文法设计占据重要地位**，为了完成更加丰富的功能，文法改进迭代了很多版本；编程实现使用了**丰富的 STL 模板类并进行自定义重载**，利用其**高效查询以及去重**的特性，非常适合高效的分析过程，逻辑更加清晰；整个编程的代码实现没有针对特定文法（除任务 1 DFA 转化后部分），换用其他文法同样有效；程序运行时的**过程信息和提示都分别实时输出在 txt 文本和 terminal 中**，体现**程序分析的完整性（文法的 V_n 、 V_t 、DFA 终态组成、状态转移表、能否推出空、First 集、项目集规范组、项目集转化关系、ACTION-GOTO 表、LR(1)分析的过程信息）**

2. 开发工具

出于个人喜好和进一步锻炼项目编程技能的考量，继 Java 软件课程开发课设后我做出了新的语言和工具选择（以下工具包括报告制作和思路整理、内容设计）

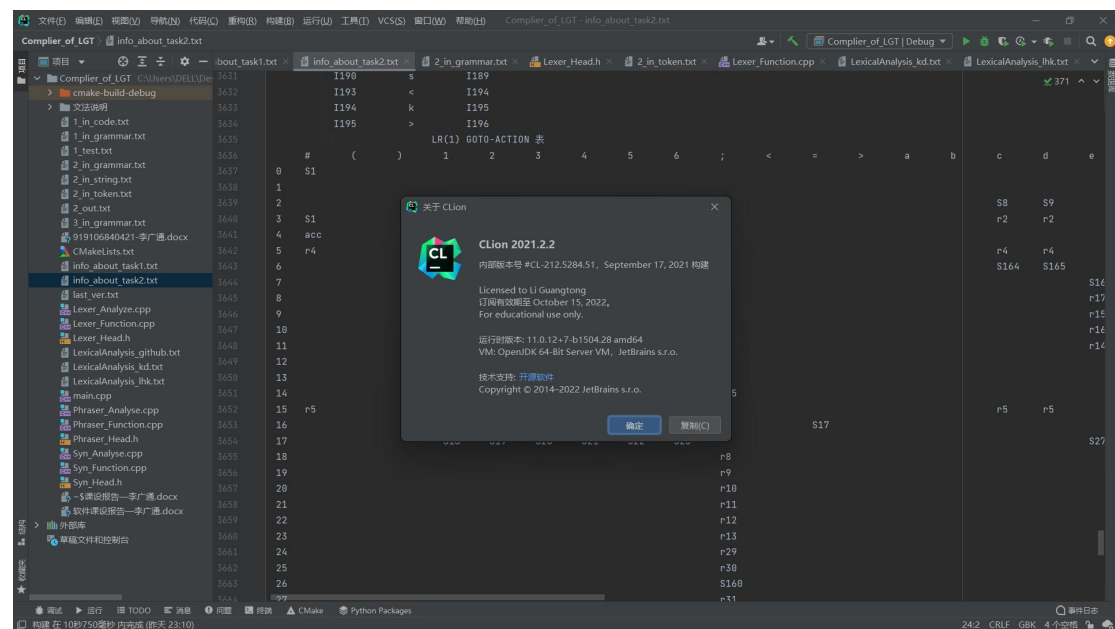
开发语言: C++

开发工具: Clion 2021.2.2

Adobe Photoshop 22.0.0

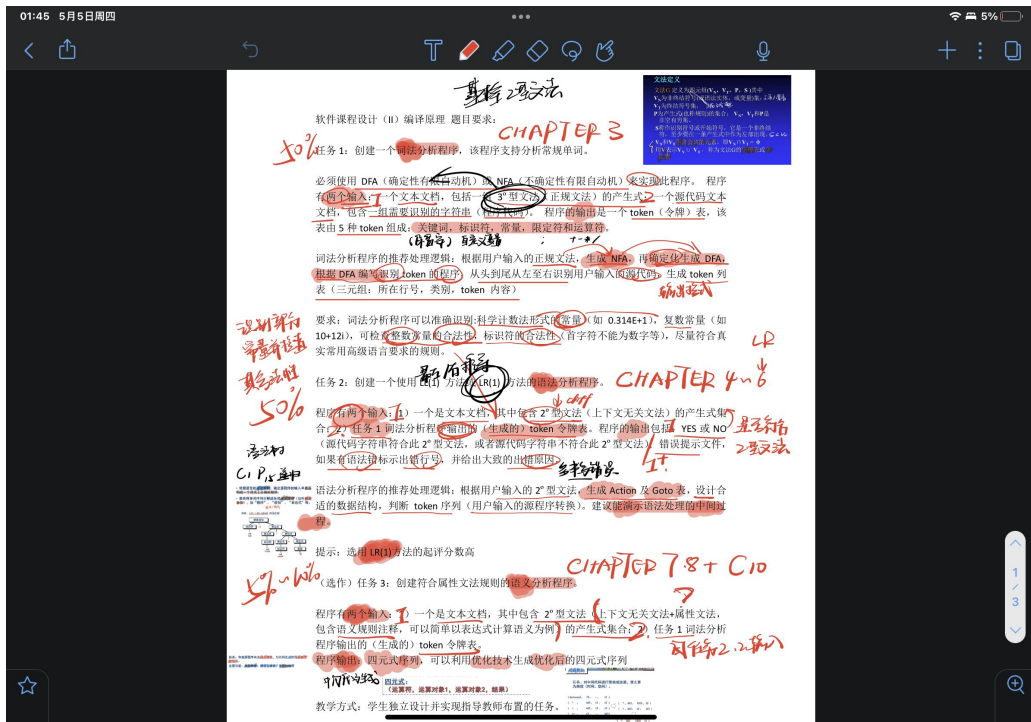
Procreate 5.2.6

Xmind 2021.0527.0001



3.开发设计思路

通过对**报告的解读**,明确理解了相关设计要求和所涉及到的编译原理相关知识之后,对上学期编译原理课件进行回顾翻阅后**总结整体设计流程**后进行各个部分的开发。



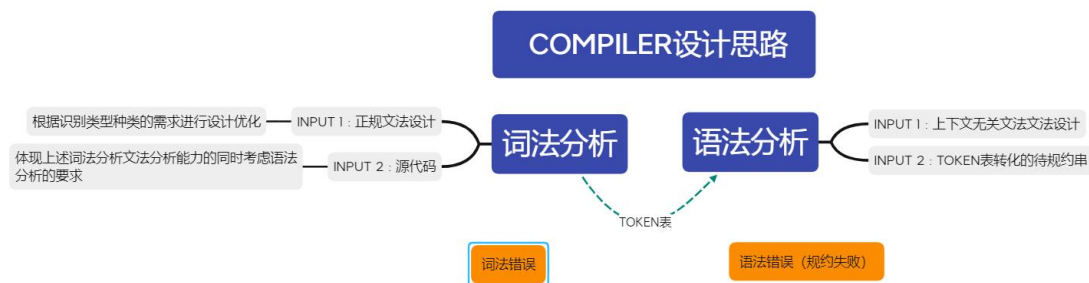
整体设计思路为：词法分析和语法分析作为编译过程中两个预处理的过程，并不涉及更深层的语法规义，但错误信息应该是全程收集的，项目开发顺序根据过程也有先后。

处理逻辑应该为：

1. **词法分析：**源程序代码读入后根据词法分析器中已经预先设计好的正规文法进行分析，这其中**先对正规文法求 NFA，再对 NFA 进行 DFA 转换**，得到 DFA 状态集合后根据 DFA 状态集中的组成部分（由哪些 NFA_Vn 组成），结合文法设计功能对单词扫描过程进行相应的限制，防止**不应出现的状态跳转冗余**。然后**逐个单词**对源代码文件进行**扫描**，针对不同的状态进行不同的 **token 分类**。

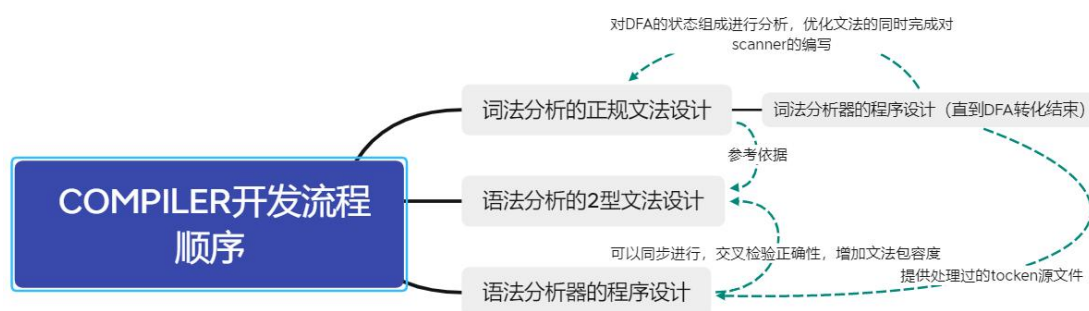
2. **语法分析：**然后进行语法分析，首先完成对预先设计好的上下文无关文法的预处理，包括**求取 First 集，根据 First 求项目集规范簇，生成新的状态转换关系后求取 ACTION-GOTO 表**；做好准备工作后应该先将词法分析生成的 token 表**转化成符合语法分析文法格式的待规约串**，再根据 ACTION-GOTO 表

一步步对状态和符号栈进行操作，通过查询决定动作（规约、移进、接受、失败）并根据失败位置对 token 表查询输入错误行号和可能的大致错误信息，整个流程应该数据可视化，完整展现过程信息。



可以看到词法分析对文法设计和编程实现有一定的先后性要求 NFA->DFA 转换后要不断交叉验证比对优化无效转换状态；而语法分析则不同，可以同时进行文法设计和编程的工作，只要注意文法对源程序的结构包容度就可以了。

具体开发流程如下：



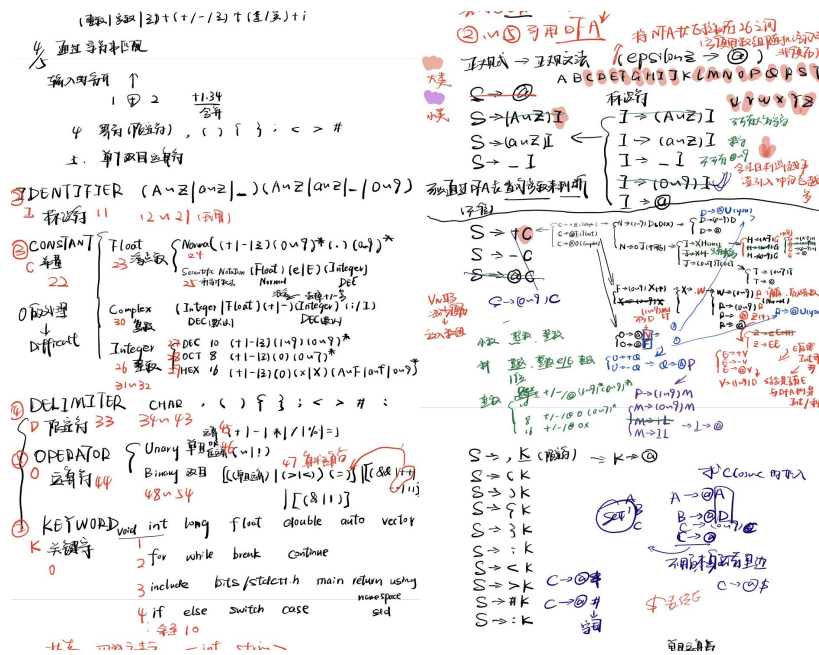
3.1 文法设计

<1>词法分析的正规文法设计

此部分要求文法为正规文法，虽然文法设计上来讲可以不设计 ϵ (epsilon) 产生式（例如 $A \rightarrow \epsilon$ ），降低编程实现的复杂度。但从功能角度来将会花费大量的时间来进行相同语义的产生式转化，导致本就复杂的产生式可读性差；且 NFA->DFA 步骤中关键的一步就是对 ϵ 弧转化后的状态集求 epsilon 闭包进行状态扩展，为了体验完整的 compiler 设计开发流程，使用带空的正规文法进行词法分析。

（其中 ϵ 在程序中用字符‘@’表示）

设计时先总结不同 token 类型的整体特征形成正规式，再结合内部分类进行细节特征的表达引入，所有类型的正规式总结设计完成后，将其转化为正规文法并尽量消除冲突就可以作为词法分析文法进行使用了。（过程如下图）



该文法可以识别的类型有：**标识符、常量、限定符、运算符、关键字**
 其中有以下细节设计：

合法性设计：

1. 标识符号只能以字母或者_开头
2. 浮点数中科学计数法 E 后必须跟整型
3. 8 进制 0 开头且组成数字必须 0~7 范围内
4. 16 进制中字母只能取 A~F
5. 浮点数小数点前不带数字不合法

...

可识别的细分数据类型：

单双目运算符、8、10、16 进制数、科学计数法、复数

文法基本信息（程序自动统计，NFA->DFA 前）：

正规式条数：210

Vn 数目：13

{ CEFHIKMOPRSWX }

Vt 数目：67

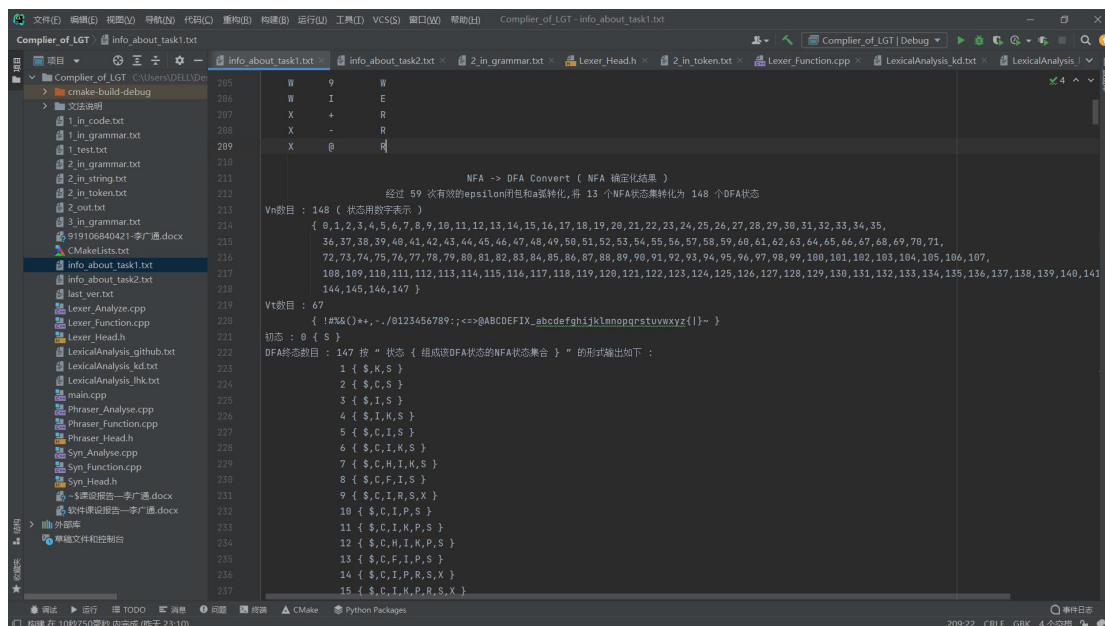
{ !#%&()*+,-./0123456789;:<=>@ABCDEFGHIX_abcdefghijklmnopqrstuvwxyz{|}~ }

初态：S

NFA 终态数目：7

{ CEFIKOP }

经过 NFA->DFA 转换后信息如下（程序自动统计，NFA->DFA 后）：



<2>语法分析的上下文无关文法设计

由于选用的是 LR(1)方法进行分析，而 **LR(1)文法**是相较 LALR (1) 等文法来讲**更加广泛包容的，且查错也更加方便**，因此节约了很多文法设计以及调试时间，语法分析文法更多地注重于如何将循环和声明、赋值、头文件引入、命名空间使用、主函数等各个结构和组成成分进行状态分类，**在容许相互嵌套的条件下又能够使用编程逻辑进行源程序的结构限制**（例如主函数不可以有两个返回值）

处理好状态嵌套和冲突之间的联系，并用合适的 2 型文法进行表达，是本步骤的关键。

由于语法分析处理的是 **token 表项之间的逻辑结构**，所以相比词法分析，**产生式要少很多，产生式及其含义如下：**

```
S->HPI // 主体三大部分 head 头文件可多个
H->O // 头文件
O->#k<k>
H->OH
P->kkk; // 命名空间
I->Ym()M // 主函数只能有一个
M->{DrK;} // 保证主函数只有一个返回值(return 常量)
K->1 // K->各种常量类型
K->2
K->3
K->4
K->5
```



```

K->6
Y->i          // 变量类型 int double float complex
Y->d
Y->f
Y->c
D->AD          // 声明 可以进行 type e; 的定义
A->Ye;
D->A
D->PD          // 变量声明的同时用表达式|变量|常量进行赋值 type e=
                { e 和 K 组成的边长表达式 | 单独的 e 或 K }
D->P
P->Ye=M;
D->BD          // 计算赋值 e= { e 和 K 组成的边长表达式 | 单独的 e 或 K }
D->B
B->e=M;
M->K;
M->e;
M->J
J->K
J->e
M->MsM          // temp 用于计算赋值和声明(同时赋值)的单目运算符引
                入(可变长)
M->sM
D->ED          // for 循环
D->E
E->f(PebK;U){D} // for 中迭代器可以是变量声明,变量声明(+赋值) 变更可
                以左或者右方双目运算符
U->eb
U->be
D->FD          // while 循环 (可以是任何常量|变量) 随时 break||continue
D->F
F->w(J){D}
D->o;          // break   a   |   continue o

```

D->a;

文法基本信息（程序自动统计）：

产生式条数：43

Vn 数目：15

{ ABDEFHIJKMOPSUY }

Vt 数目：28

{ #()123456;<=>abcdefikmorsw{} }

各 Vn 能否推出空：

P	O	M	K	J	I	S	B	A	Y	H	U	D	E	F
N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

各个 Vn 的 First 集合：

First (A) = { c,d,f,i }

First (B) = { e }

First (D) = { a,c,d,e,f,i,k,o,w }

First (E) = { f }

First (F) = { w }

First (H) = { # }

First (I) = { c,d,f,i }

First (J) = { 1,2,3,4,5,6,e }

First (K) = { 1,2,3,4,5,6 }

First (M) = { e,s,{ }

First (O) = { # }

First (P) = { c,d,f,i,k }

First (S) = { # }

First (U) = { b,e }

First (Y) = { c,d,f,i }

项目集规范族构建结果：197 个

3.2 编程实现

对整个开发流程有了具体思路之后，发现本次的**软件课程设计有很鲜明的特征：无论是词法还是文法分析，无论对状态集还是项目集，大大小小的数据结构增删查改都涉及到表项重复的问题**（而针对庞大的词法分析产生式数量，DFA_move 线性矩阵的表项已经上万了），而对表项的查找通过遍历或者多维数组下标转化的形式还是存在一定问题。的确可以使用大数组用空间换取时间，但是在**各种数据结构大小未知**的情况下不断试错，找到合适的数组大小浪费时间不说，查找起来**遍历的时间复杂度 $O(n)$** 也是很慢的。

为了**兼顾快速查询，并减少是否重复的判断函数操作**，编程过程中的**数据结构大量使用了 STL 模板类中的 set 和 map ($O(\ln N)$)、unordered_map ($O(1)$)**通过比较函数定义和运算符重载，使得严格弱序成立，自定义类型的自动去重排序为简明的数据结构和代码开发逻辑带来了极大便利。

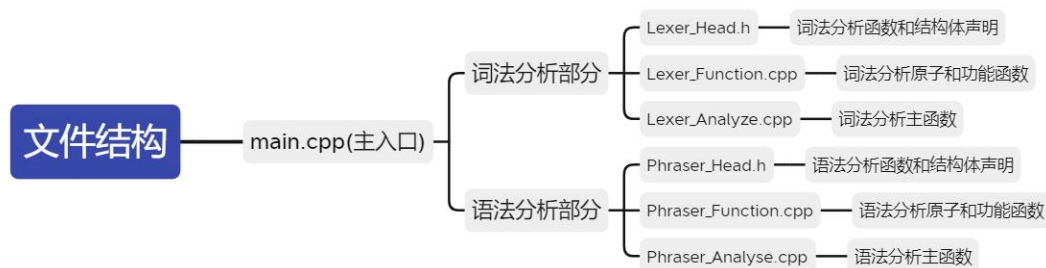
这一数据结构的使用为编程简易性带来的影响是巨大的：

遍历方便、查询方便、插入方便、按序输出方便

不过要时刻注意.find()返回是否是尾部迭代器地址的问题，处理空类的问题层出不穷，还需要对各种运算符重载，但毕竟鱼与熊掌不可兼得，对我而言 STL 模板类的大量应用为我的开发和数据结构设计带来了帮助。

3.2.1 文件结构

为了文件结构的清晰，将头文件和源程序、函数部分分离，使用标准的项目开发结构，将数据结构和函数声明全部放在头文件中，任务 1 和 2 的全局变量使用头文件配合 extern 进行引用，方便数据共享，而原子和功能函数在_Functinon 命名的源文件中定义，全部封装到 Lexer_Analyze 和 Phraser_Analyse 程序中。Main 函数只需要使用词法和语法分析的两个函数就可以实现全部功能。



3.2.2 数据结构、主要函数说明

<1>数据结构说明

词法分析器：

使用各种 set 对状态集，NFA 状态转移矩阵，DFA 状态转移矩阵等一系列多个重要数据结构进行设计并进行 typedef，**重载 operator () 使其自动去重。**

```
// 结构体及类型定义
typedef struct NFASET{...}NSET; // NFA 状态集 || 可用于组成nfa_move矩阵的item(if_finalstate无效)
using turnto = std::pair<char,char>;
namespace std {...}

typedef pair<pair<char,char>,NSET> nfa_move_item; // NFA_movetable 中的单元 根据pair<状态,VT> 状态(VT)->NSET(状态集)
typedef pair<pair<int,char>,int> dfa_move_item;
struct cmp{...}; // 结构体比较函数——用于set去重

typedef set<pair<int,bool>,cmp> temp_for_print; // 用于有序输出
typedef set<nfa_move_item ,cmp> NFA_movetable;
typedef set<dfa_move_item ,cmp> DFA_movetable;
typedef set<pair<int,set<char>> ,cmp> NEW_of_OLDstate; // 纪录新状态的组成部分(旧状态) first 不能取char (不能保证唯一性)
typedef unordered_map<turnto,set<char>> SEARCH_NFA; // 将move_nfa_matrix_final 转化成可以用下标访问的结构

int VT_NUM=0; //84 终结符个数不变
int NFA_VN_NUM=0; //33
int DFA_VN_NUM=0;
char start_state;
int DFA_start_state=0;
int pv=0; // 动态纪录epsilon闭包和a弧转化 次数
int error_number=0;
char final_bef='S',final_new;
set<char> VT; //NFA终结符
set<char> VN; //NFA非终结符
set<int> DFA_VN; // DFA非终结符 无效设计, set自排序可能不同状态开头代表元素也相同
NEW_of_OLDstate 0_IN_NEW; // 真正的DFA非终结符结构设计
set<char> nfa_final_state;
map<int,pair<pair<int,string>,char>> for_task2; // 根据字符号查询行号\位置; 字符index, <行号, 字符>原本类型
unordered_map<char,set<char>> VN_CLOSURE_SET; // 终结符的闭包
unordered_map<int ,set<char>> DFA_FINAL_STATE_FOR_PRINT; // DFA的终态map 数字号-原组成状态 由dfa_final_state 和 0_IN_NEW生成
unordered_map<int ,set<char>> DFA_NOTFINAL_STATE_FOR_PRINT; // 同上
map<set<char>,int> dfa_state_search; // 辅助提供通过nfa状态集合寻找dfa状态的可能 和0_IN_NEW 互反
unordered_map<char,set<char>> useable_vt_for_NFA_VN; // 处理收集nfa中单个vn可用的vt
unordered_map<int,bool> dfa_state_ifcircleandepsilon; // 当前状态是否已经经过了1弧转换和词法语法分析
unordered_map<int,bool> dfa_final_state; // DFA的状态是否为终结态 (!!!!!!!!!! 还是人为规定转换终态只要有$就算) 在0_IN_NEW中
int dfa_vn_counter=0; // 用于dfa状态计数插入 输出记得char int转换
int for_task2_num=1;
int line_index_for2=0;
NFA_movetable move_matrix_NFA; // for auto 只能遍历而不能插入
NFA_movetable move_matrix_NFA_final; // 真正的集合形式
SEARCH_NFA move_matrix_NFA_MAP; // 从final转化而来
DFA_movetable move_matrix_DFA;
map<pair<int ,char>,int > move_matrix_DFA_FOR_SEARCH; // 由move_matrix_DFA演化而来, 方便进行dfa寻路, 词法分析
```

语法分析器：

设计思路同词法分析器，增加了更多的自定义类内操作，进一步提升代码复用性。

```
// 结构体及类型定义
typedef struct right_source{...}rs; // 用于储存产生式右部

typedef struct project_set_item {...}project_set_ITEM; //项目集中的单条产生式（包含各种接收和tail信息） 可以直接==判等
typedef struct action_item{...}ACTION_item; // 用于ACTION表的构建second
struct compare{...};

typedef set<rs,compare> RIGHT_SET; // 存放一个vn对应所有的右部
typedef set<project_set_ITEM,compare> PROJECT_SET; // 项目集 项目默认按照输入时的顺序进行排序 使用if_equal判等
set<char> VT_2;
set<char> VN_2; // 不包括@
set<char> VN_FOR_TITLE; // GOTO 表头
vector<char> chars_for_analyse; // 用于存储要移进—规约的字符串
unordered_map<char,bool> VN_ifcanbe_epsilon; // 各个vn是否可以推出空 GOAL 1
map<int,pair<char,string>> PRODUCTION_FOR_ACTION; // 用于action表进行规约时根据行号查找规约到哪个vn
map<char,RIGHT_SET> PRODUCTION_SOURCE; // init将正规式存入的形式
map<char,set<char>> VN_FIRST; // VN的first集
map<char,bool> VN_getfirst_ifdone; // 各个是否已经完成求取first集的操作
map<char,set<int>> line_search_for_closure; // 统计同一vn为左部的产生式行号
set<pair<pair<int,char>,int>,compare> MOVE_OF_NEW_STATE; // 纪录新状态的转化matrix
set<pair<pair<int,char>,int>,compare> GOTO_TABLE; // GOTO表
set<pair<pair<int,char>,ACTION_item>,compare> ACTION_TABLE; // ACTION表
map<pair<int ,char>,int > goto_for_search; // 用于打印和查询扫描
map<pair<int ,char>,ACTION_item > action_for_search;
unordered_map<int,PROJECT_SET> NEW_STATE; // 用数字做index的新状态
int STATE_NUM=0; // 状态从0开始计数
extern map<int,pair<pair<int,string>,char>> for_task2; // 根据字符号查询行号\位置；字符index，<行号，字符>

ofstream output_info_2; // 输出文法分析的具体信息
ofstream output_2; // 输出yes|no 并给出错误行号和提示信息
```

<2>主要函数说明

词法分析器：

1. void initialization ();

对各种常量数据类型进行初始化，并打开词法分析要输入输出的文件流

```
void initialization () {
    VT_NUM=0;
    NFA_VN_NUM=0;
    DFA_VN_NUM=0;
    dfa_vn_counter=0;
    VT.clear();
    VN.clear();
    start_state=' ';
    nfa_final_state.clear();
    dfa_final_state.clear();
    VN_CLOSURE_SET.clear();
    DFA_NOTFINAL_STATE_FOR_PRINT.clear();
    DFA_FINAL_STATE_FOR_PRINT.clear();
    dfa_state_search.clear();
    move_matrix_NFA.clear();
    move_matrix_NFA_final.clear();
    move_matrix_NFA_MAP.clear();
    move_matrix_DFA.clear();

    source_code= fopen( _Filename: "../1_in_code.txt", _Mode: "r+"); // 打开要分析的源代码文件，用于scanner函数进行词法分析
    grammar_1=fopen( _Filename: "../1_in_grammar.txt", _Mode: "r+"); // 打开文法流 LexicalAnalysis_lhk
    // grammar_1=fopen("../1_test.txt","r+");
    output_string.open( s: "../2_in_string.txt");
    output.open( s: "../2_in_token.txt"); // 输出token表
    output_info.open( s: "../info_about_task1.txt"); // 输出文法转化和token识别相关信息
}
```

2. void Grammar_to_NFA ();

3 型文法 -> NFA (状态转移矩阵生成) 读入文法

```
void Grammar_to_NFA () {
    int n;
    char line[100]; // 使用fgets必须先分配内存，不可空指针
    // line.resize(7); // 分配内存后才能用fscanf 尽量不用resize(变capacity)
    char state_left, state_right;
    char vt;
    fseek(grammar_1, _Offset: 0, SEEK_SET);
    fgets(line, _MaxCount: 5, grammar_1);
    string numoffline(line);
    n= stoi(numoffline);
    output_info<<"词法分析文法 相关信息统计(已排序处理) "<<endl;
    output_info<<"正规式条数 : "<<n<<endl;
    for(i, a: 0, n){ // less ABKY IV
        // fscanf(grammar_1, "%s", &line[0]);
        fgets(line, _MaxCount: 100, grammar_1);
        state_left=line[0];
        vt=line[3];
        if(i==0){ // 初态设置
            start_state=state_left;
        }
        if(!if_inVN(state_left, VN: VN)) // 对左部进行添加
            VN.insert(state_left);
        if(!if_inVT(vt, VT: VT)) // 对右部非终结符进行添加
            VT.insert(vt);
        if(((int)line[4]>=(int)'A'&&(int)line[4]<=(int)'Z')||((int)line[4]>=(int)'a'&&(int)line[4]<=(int)'f')){
            state_right=line[4];
        }
        else{ // 无VN 对左边部分添加 (vt)->#(终态)
        }
    }
}
```


3. void NFA_to_DFA ();

NFA 确定化，生成新状态转移矩阵，是整个词法分析中最重要的函数，首先对初态求闭包，建立一个栈，将初态压入 stack 中，对栈顶元素求取所有可能的 a 弧转换并取闭包，如果是新状态集则作为 DFA 的新状态存入相应数据结构，并压栈，否则只对 DFA 的 move 矩阵添加表项就推出本轮循环，直到 stack 为空，NFA->DFA 转化结束。

```
void NFA_to_DFA () {
    NSET pre_set;
    pre_set.set_member.insert(start_state); // 先处理初态
    stack<NSET> s;
    get_closure(&pre_set, &move_matrix_NFA); // !!!!!!!wait for write 如遇到->@ 后集合中有$
    s.push(pre_set);
    // for(auto i:pre_set.set_member)
    //     cout<<i;
    // cout<<endl;
    DFA_start_state=0; // 0状态是起始状态
    add_NSET_toState(&pre_set, 0_IN_NEW, &dfa_state_search); // 得到旧状态组成的新状态结构体，方便后续操作 (将NSET变成M)
    pu=1; // 当前行第几个
    set<char> temp_allA_inSET; // 状态集中所有状态可以进行的a集合(剪枝，大幅优化计算时间，对多状态计算效果显著)
    map<set<char>,int> cut_tree; // 同一行的如新增状态相同，则不用再二次取闭包并添加状态了，直接退出即可
    set<char> add_state; // 当前行状态经过i弧转化后新增的状态
    map<char,char> for_quickcompare;
    map<int,bool> have_gothrough_line_states; // 一个状态是否已经走过完整的一行了
    have_gothrough_line_states.clear();

    int counter=0;

    while(!s.empty()){
        pre_set=s.top();
        s.pop();
        int bef= if_inNewSet(&pre_set, dfa_state_search: dfa_state_search); // 一行的起始状态不变
        if(have_gothrough_line_states.find(bef)!=have_gothrough_line_states.end())
            continue;
        pu=1;
    }
}
```

4. void scanner ();

扫描代码源文件进行词法分析,生成 token 表放入 2_in_token.txt

```
void scanner () {
    ifstream source("1_in_code.txt");
    string line;
    vector<string> words;
    map<pair<int ,char>,int> DFA_SEARCH; // 方便dfa查询
    bool iffIRSTxiaoz=true,iffIRSTdayu=true;
    DFA_SEARCH.clear();
    for(auto ok:move_matrix_DFA)
        DFA_SEARCH.insert({ok.first,ok.second});
    output<<"          TOKEN TABLE"<<endl;
    output<<"          line          "<<" type          " <<" word          "<<endl;
    // while(getline(source,line)){
    //     for(auto i:line)
    //         cout<<i<<" ";
    //     cout<<endl;
    // }
    cout<<endl<<"          task 1 is running ... "<<endl<<endl;
    cout<<"源代码分析结果如下所示 : "<<endl;

    while(getline(&source, &line)) {
        line_index_for2++;
        words.clear();
        split(line, line, &words);
        for(auto i:words){
            cout<<i<<" ";
        }
        cout<<endl;
    }
}
```

语法分析器：

1. void vn_epsilon_process ();

根据 task2 文法处理 Vn 判断是否可以推出 epsilon

```
void vn_epsilon_process (){ // 要求vn必须是大写英文字母
    set<char> VN_can_epsilon, VN_not_epsilon, VN_unknow; // 先求出定能|不能求出闭包的状态;
    bool all_have_vt=true;
    bool if_added_to_must=false; // 是否已加入前两个绝对集合
    int counter=0;
    for(auto i:PRODUCTION_SOURCE){
        counter=0;
        if_added_to_must= false;
        for(auto j:i.second){
            if(j.if_a_single_epsilon){ // if have ->@
                if_added_to_must=true;
                VN_can_epsilon.insert(i.first);
                goto here;
            }
            for(auto k:j.right_inorder){ // 如果一个vn的所有产生式的右部的都含有vn, 必不能产生空
                if(j.right_for_compare[0]<'A' || j.right_for_compare[0]>'Z'){
                    counter++;
                    break;
                } // 如果右部中存在终结符
            }
        }
        if(counter==i.second.size()){
            VN_not_epsilon.insert(i.first);
            if_added_to_must=true;
        }
        if(!if_added_to_must)
            VN_unknow.insert(i.first);
        here : ;
    }
    bool ifhave_added=false;
    while ( !VN_unknow.empty() ){
```

2. void vn_get_FIRST_SET ();

求取 Vn 的 first 集

```
void vn_get_FIRST_SET (){
    map<char, set<char>> first_temp;
    int counter=0; // 统计 对每一个vn 的所有产生式右部 满足第一个字符为vt|@的个数
    for(auto i:VN_2){ // 对每个vn
        counter=0;
        for(auto j:PRODUCTION_SOURCE.find(i)->second){
            if(j.if_a_single_epsilon){ // 如果有->@
                if(first_temp.find(i)!=first_temp.end()){
                    first_temp.find(i)->second.insert( x: '@' );
                }
            }
            else{
                set<char> tt;
                tt.insert( x: '@' );
                first_temp.insert( x: {i, tt} );
            }
            counter++;
            continue;
        }
        if(j.right_for_compare[0]<'A' || j.right_for_compare[0]>'Z'){ // 如果右部第一个为vt
            if(first_temp.find(i)!=first_temp.end()){
                first_temp.find(i)->second.insert( x: j.right_for_compare[0] );
            }
        }
        else{
```

3. void LR_1_CREATE ();

重要函数，用于求取 LR 项目集合的同时对,新状态转换表进行添加（增广文法 $\$ \rightarrow S$ ）编程思路和词法分析中 NFA_to_DFA 一致，都是使用 stack 进行项目集求取。

```
void LR_1_CREATE () {
    // $->S 为第0条 增广文法的加入引发一系列变更
    VT_2.insert(x: '#'); // error reason
    if(VN_ifcanbe_epsilon.find(x: 'S')->second)
        VN_ifcanbe_epsilon.insert(x: {x: '$', y: true});
    else
        VN_ifcanbe_epsilon.insert(x: {x: '$', y: false});
    PRODUCTION_FOR_ACTION.insert(x: {x: 0, y: {x: '$', y: "S"}});
    rs the_begin( right: "$->S");
    RIGHT_SET first_temp;
    first_temp.insert(the_begin);
    PRODUCTION_SOURCE.insert(x: {x: '$', first_temp});
    VN_FIRST.insert(x: {x: '$', VN_FIRST.find(x: 'S')->second});
    // 建立第一个项目集
    PROJECT_SET pre_set;
    pre_set.insert(x: init_project_set_item( line_index: 0));
    get_CLOSURE( &: pre_set); // 取CLOSURE
    NEW_STATE.insert(x: {STATE_NUM++, pre_set}); // 加入新状态
    stack<PROJECT_SET> s;
    s.push(pre_set);
    set<char> vt_or_vn_forconvert; // 用于GOTO的可能字符
    while(!s.empty()){
        pre_set=s.top();
        s.pop();
        int pre= if_in_new_state( aft_set: pre_set),aft; // 用于辅助LR (1) 状态转化表进行插入
        vt_or_vn_forconvert.clear();
        for(auto i:pre_set){ // 查找所有.的右部终结和非终结符
            for (int j = 0; j < i.right.size(); ++j) {
                if(i.right[j]=='.'){
                    if(j+1<i.right.size()){

```

4. void get_ACTION_and_GOTO ();

根据 ppt 中的规则和含义对新的项目集合转化关系进行统计得到 ACTION 和 GOTO 表

```
void get_ACTION_and_GOTO () {
    // 先对goto表添加
    for(auto i:MOVE_OF_NEW_STATE){
        if(VN_2.count(i.first.second)) // MOVE_OF_NEW_STATE中character为Vn直接添加GOTO表
            GOTO_TABLE.insert({i.first,i.second});
    }
    // 再对action表添加
    // S r acc添加
    for(auto i:NEW_STATE){
        for(auto j:i.second){
            if(*j.right.rbegin()=='.'){ // r和acc情况
                if(j.tail=='#'&&j.left_Vn=='$'&&j.right.begin()=='S'){ // acc情况
                    ACTION_item tempt( movement_first_char: 'a', number: 0);
                    if(!if_is_new_ACTION_item( temp: {x: {i.first, y: '#'},tempt}))
                        ACTION_TABLE.insert(x: {x: {i.first, y: '#'},tempt});
                    continue;
                }
            }
            if(VT_2.count(j.tail)){ // r情况

```

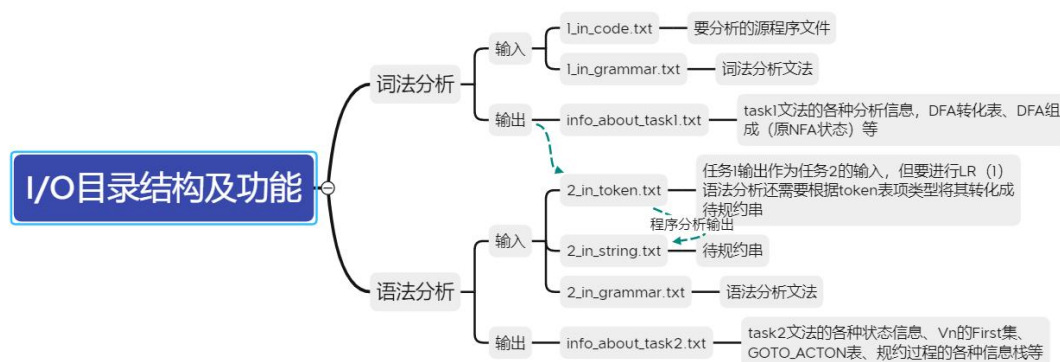
5. void LR_1_ANALYSIS ();

LR(1)分析，扫描返回结果 YES 或者 NO，如果规约失败，输出错误行号和可能的错误信息

```
void LR_1_ANALYSIS (){
    ifstream input_string( s: "../2_in_string.txt");
    string temp;
    bool if_accept_success=false;    // 是否被成功接受
    getline( &: input_string, &: temp);
    for (int i = 0; i < temp.length(); ++i)    // 将待分析字符串读入
        chars_for_analyse.push_back(temp[i]);
    chars_for_analyse.push_back('#');
    cout<<"移进规约分析过程详见 info_about_task2.txt ( 实时生成 )" <<endl<<endl;
    // step 1 单独处理
    stack<int> state;    // 状态栈
    stack<char> character;    // 符号栈
    vector<int> state_fornow;    // 当前行使用方便查询和输出
    vector<char> character_fornow;
    state.push( x: 0);
    character.push( x: '#');
    int step_num=0;    // 步骤号
    int char_for_access=0;
    output_info_2<<"                LR(1) 规约分析过程展示" <<endl;
    while(!if_accept_success){    // 注意引入无表项时的break
        output_info_2<<step_num++<<" ";
        if(step_num==2)
            ;
        get_stack_state_vector_version( state: state, &: state_fornow);
        get_stack_char_vector_version( character: character, &: character_fornow);
        for(auto i:character_fornow)    // 输出当前符号栈
            output_info_2<<i;
        output_info_2<<" ";
        for(auto i:state_fornow)    // 输出当前状态栈
            output_info_2<<i<<" ";
    }
}
```

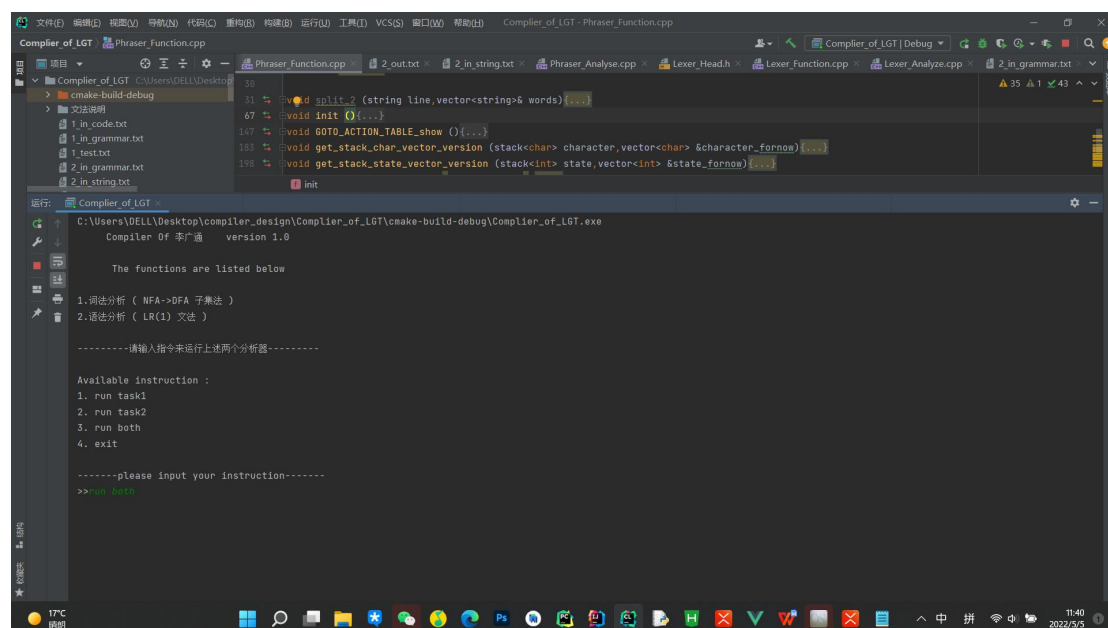
4.实例演示和效果说明

本程序的提示信息和对**文法**的分析过程，**错误提示信息**都十分友好，**terminal 中输出关键信息**（主要是实验要求中的信息和错误提示），而需要占用大量篇幅的文法分析结果、规约过程信息、状态转移矩阵、GOTO—ACTION表等受限与篇幅**分任务实时输出到不同的文本文件中**，其中**输入输出对应关系和文本文件内容结构**如下：



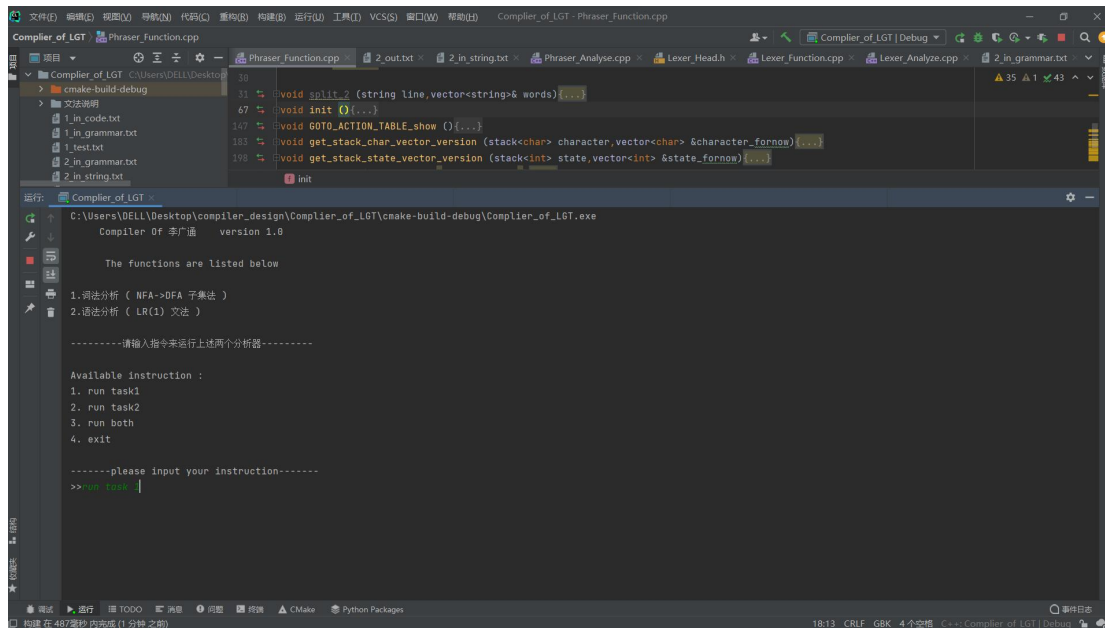
此项目在实现课程设计的基础上还实现了**各种过程分析的输出统计**、方便实现**全流程的实时分析**

主程序控制台中根据程序自定义的指令输入可以实现任务 1,2 的单独运行，也可以顺序执行，大大方便了根据错误信息对要识别的源代码进行实时更改并验证结果。

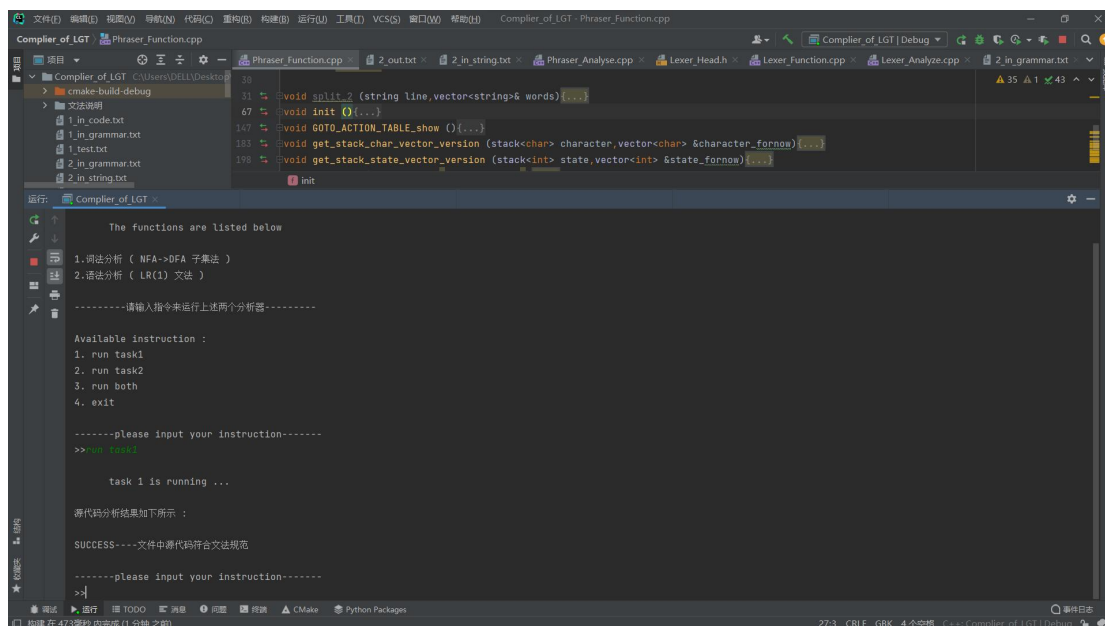


4.1 词法分析

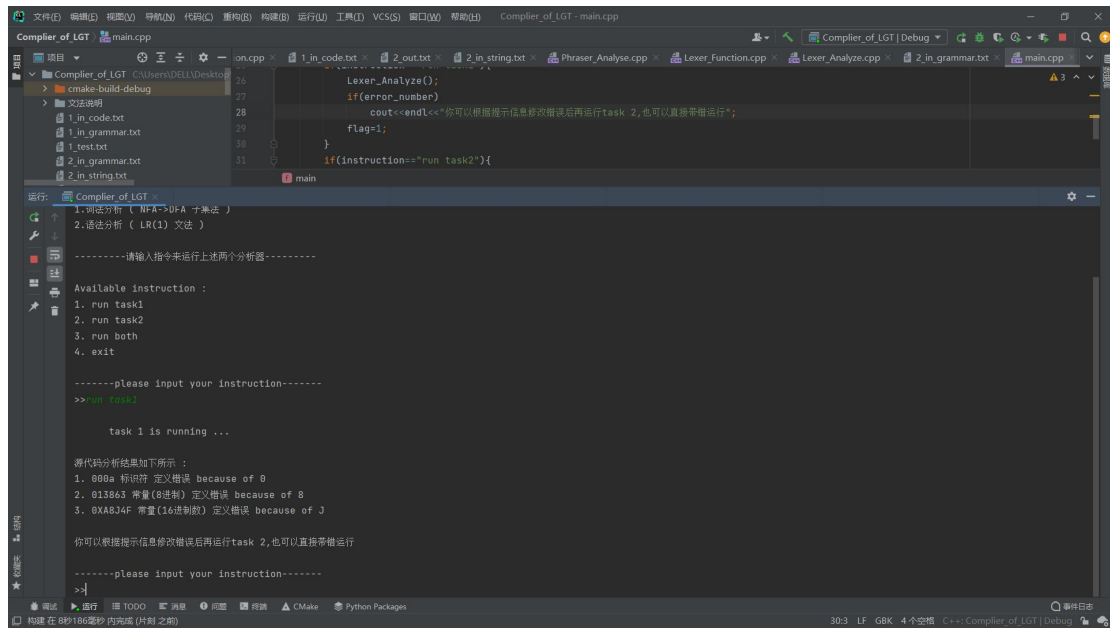
在控制台中按提示输入指令 `run task1`，运行词法分析程序



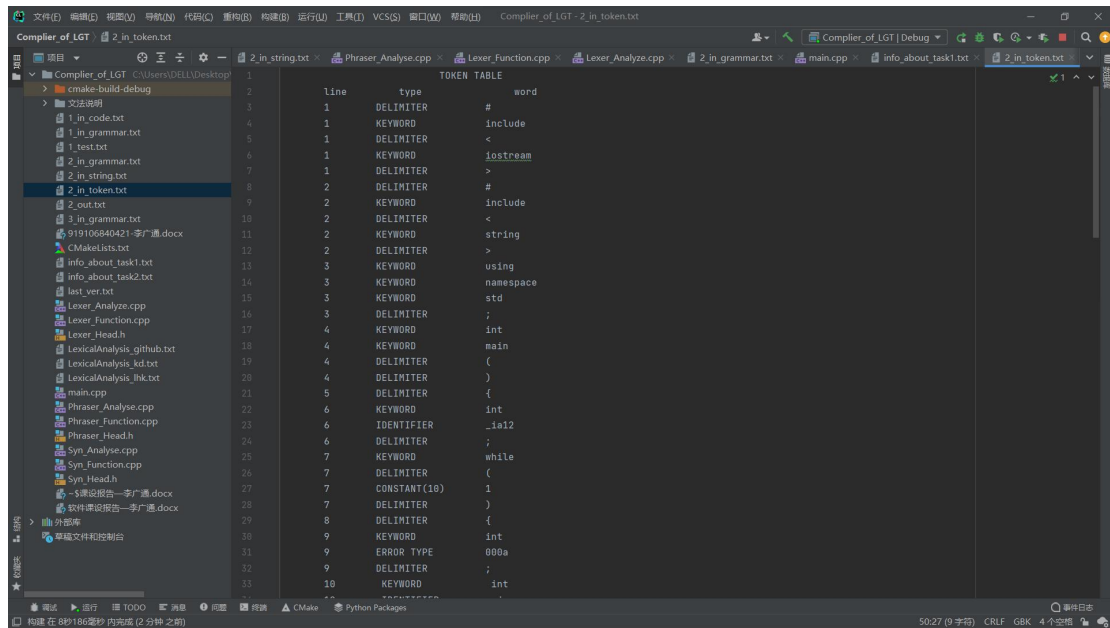
如源程序中所有单词都符合词法分析正规文法, 返回正确提示信息



如源程序有错误，无法全部识别，分条格式化输出错误信息和错误内容



输出结果 token 表在对应文本文件中：

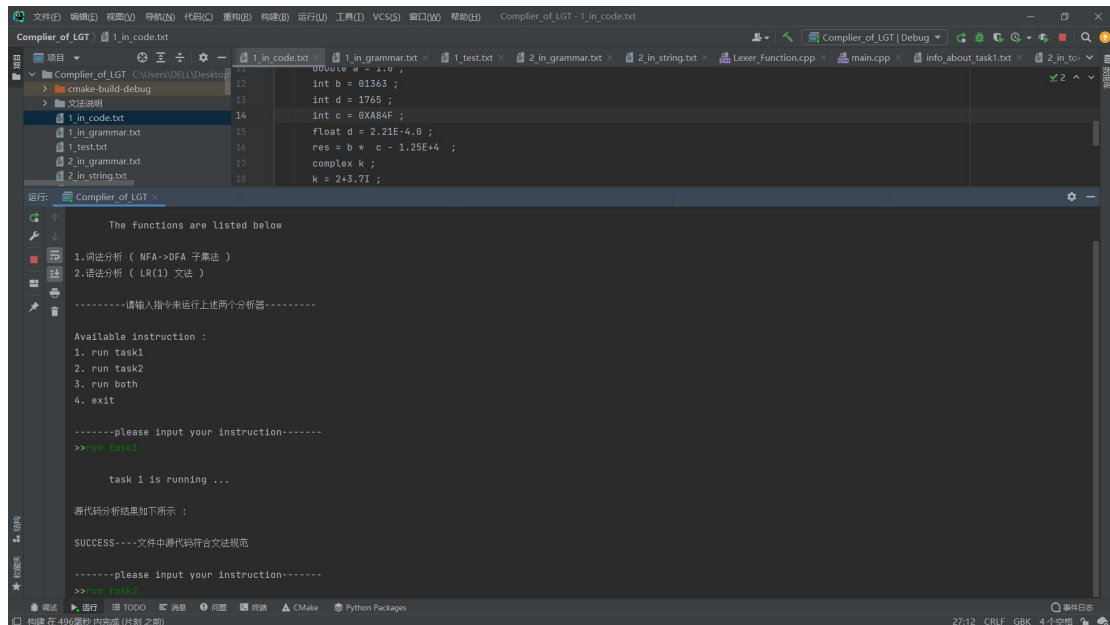


词法分析相关信息统计在 info_about_task1.txt 中



4.2 语法分析

在控制台中按提示输入指令 **run task2**，运行词法分析程序



```
Compiler of LGT - 1_in_code.txt
项目
  Compiler of LGT
  cmake-build-debug
  文法说明
  1_in_code.txt
  1_in_grammar.txt
  1_test.txt
  2_in_grammar.txt
  2_in_string.txt
  1_in_code.txt
  1_in_grammar.txt
  1_test.txt
  2_in_grammar.txt
  2_in_string.txt
  Lexex_Function.cpp
  main.cpp
  info_about_task1.txt
  2_in_to...

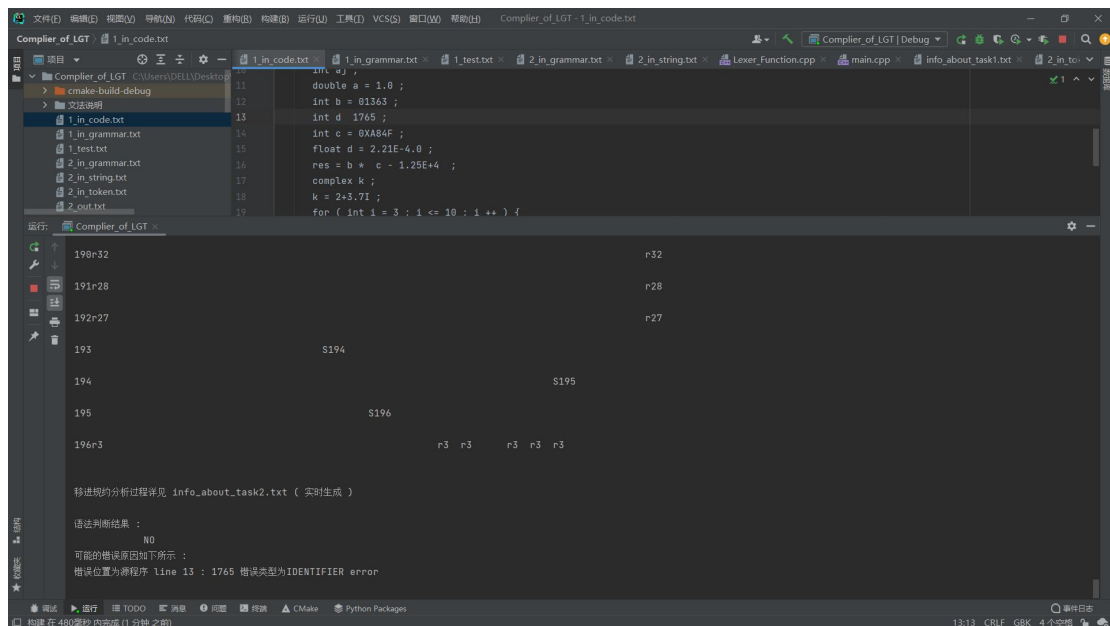
运行: Compiler of LGT
The Functions are listed below
1. 词法分析 ( NFA->DFA 子集法 )
2. 语法分析 ( LR(1) 文法 )
-----请输入指令来运行上述两个分析器-----
Available instruction :
1. run task1
2. run task2
3. run both
4. exit
-----please input your instruction-----
>>run task2

task 1 is running ...

源代码分析结果如下所示 :
SUCCESS-----文件中源代码符合文法规范

-----please input your instruction-----
>>run task2
```

如出错则返回 NO,对应错误行号和相关信息



```
Compiler of LGT - 1_in_code.txt
项目
  Compiler of LGT
  cmake-build-debug
  文法说明
  1_in_code.txt
  1_in_grammar.txt
  1_test.txt
  2_in_grammar.txt
  2_in_string.txt
  2_in_token.txt
  2_out.txt
  1_in_code.txt
  1_in_grammar.txt
  1_test.txt
  2_in_grammar.txt
  2_in_string.txt
  2_in_token.txt
  2_out.txt
  Lexex_Function.cpp
  main.cpp
  info_about_task1.txt
  2_in_to...

运行: Compiler of LGT
190r32 r32
191r28 r28
192r27 r27
193 S194
194 S195
195 S196
196r3 r3 r3 r3 r3

移进规约分析过程详见 info_about_task2.txt ( 实时生成 )

语法判断结果 :
NO
可能的错误原因如下所示 :
错误位置为源程序 Line 13 : 1765 错误类型为IDENTIFIER error

-----please input your instruction-----
>>
```

如正确规约则返回 YES,和相关信息



```
Compiler of LGT - 1_in_code.txt
项目
  Compiler of LGT
  cmake-build-debug
  文法说明
  1_in_code.txt
  1_in_grammar.txt
  1_test.txt
  2_in_grammar.txt
  2_in_string.txt
  2_in_token.txt
  2_out.txt
  1_in_code.txt
  1_in_grammar.txt
  1_test.txt
  2_in_grammar.txt
  2_in_string.txt
  2_in_token.txt
  2_out.txt
  Lexex_Function.cpp
  main.cpp
  info_about_task1.txt
  2_in_to...

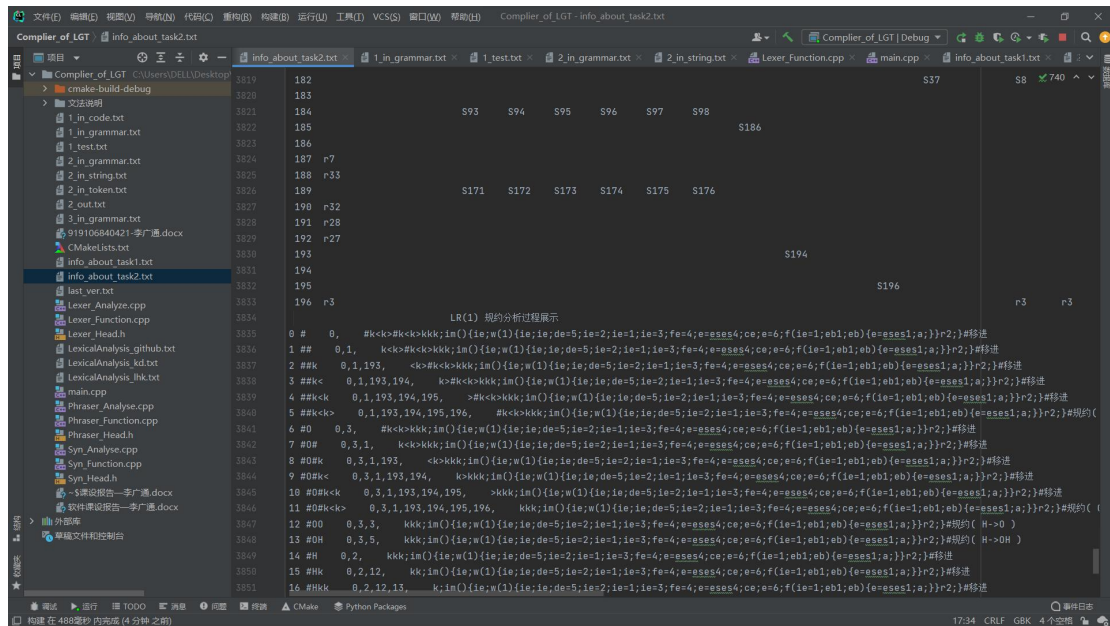
运行: Compiler of LGT
193 S194
194 S195
195 S196
196r3 r3 r3 r3 r3

移进规约分析过程详见 info_about_task2.txt ( 实时生成 )

语法判断结果 :
YES , 符合设定文法

-----please input your instruction-----
>>
```

语法分析相关信息统计在 info_about_task1.txt 中, 包括文法统计, First 集, 项目集合规范组, 项目集转化关系, LR(1)GOTO-ACTION 表, 规约分析过程展示 (规约过程显示使用的产生式和各个步骤)



5.心得体会

本次软件课程设计（II）课程对我个人而言收获很多，经历了上学期编译原理的学习，本学期的 Compiler 课设让我有了更加丰富的实践经历，对计算机体系的软件结构和编译流程有了更加深入的认识。

至于本次课设，整整一周的高强度开发，代码量 2k 多行，还要进行文法的设计，保证在尽可能少的状态数下实现尽可能多的功能实现并增加文法的健壮性，不仅是对文法设计能力的考察，对数据结构设计以及编程能力来说都是一项挑战。设计和开发过程中也不是一帆风顺的，过程中遇到过已经数不清的问题，包括 unordered_map 的绝对弱序的问题，迭代器查找判空问题，访问越界问题更是已经解决了不知道有多少次了，甚至连文法都出现过 NFA->DFA 转化不成功，stack 跑了一晚上都不为空，不过经过悉心的调试优化并需求相应解决方案，全部都得到解决。

这次的软件课程设计更多的锻炼了我对项目开发的整体把控能力，先整理出设计思路（老师的任务说明文档起到的很大帮助），理解相应设计步骤后将功能划分为大的函数并预先设计数据结构方便存取插入并自动去重，伴随着文法设计不断对其功能函数进行编写，最后在主函数中将主体函数进行调用就完成了编程工作。开发过程中对注释的编写也无处不在，方便进行代码回溯和优化，对项目的整体把控能力也得到了提升。希望下一次的软件课程设计也能得到更多的个人提升。

衷心地感谢老师能看完我的报告，**祝老师五一假期快乐，工作顺利！**