**项目测试报告**

**任务三**

**一、项目任务一测试**

**1、准备工作**

**minic的正则表达式：**

|  |
| --- |
| else|if|int|float|double|return|void|do|while  \+\ | - | \\*\ | / | % | < | <= | >= | > | ==| != | = | ; | , | \(\ | \)\ | \[\ | \]\ | { | }  PLUS | MINUS | MULTIPLY | DIVIDE | MOD | LT| LTEQ | RTEQ | RT | EQ | NE | ASSIGN | SEMI | DOU | LLM | RLM | LMM | RMM | LBM | RBM  (\_|letter)(\_|letter|num)\*  num num\* (.num num \*)?  //~\n |

**sample.tny文件：**

|  |
| --- |
| //Sample program  //In MiniC language-computes factorial  //}  int x; // an integer  int fact;  double y;  int arr[100];  void test(int a, int b)  {  a = 1;  b = 2;  return ;  }  void main(void)  {  x = 2;  y = 3;  if ( x > 1 ) //don't compute if x <= 0  do  fact = fact \* x;  while (fact >= 1);  else  x = 1;  return ; //return void  } |

**输入正则表达式：**



**2、NFA**



**3、DFA图**



**4、DFA图最小化**



**5、生成词法程序**

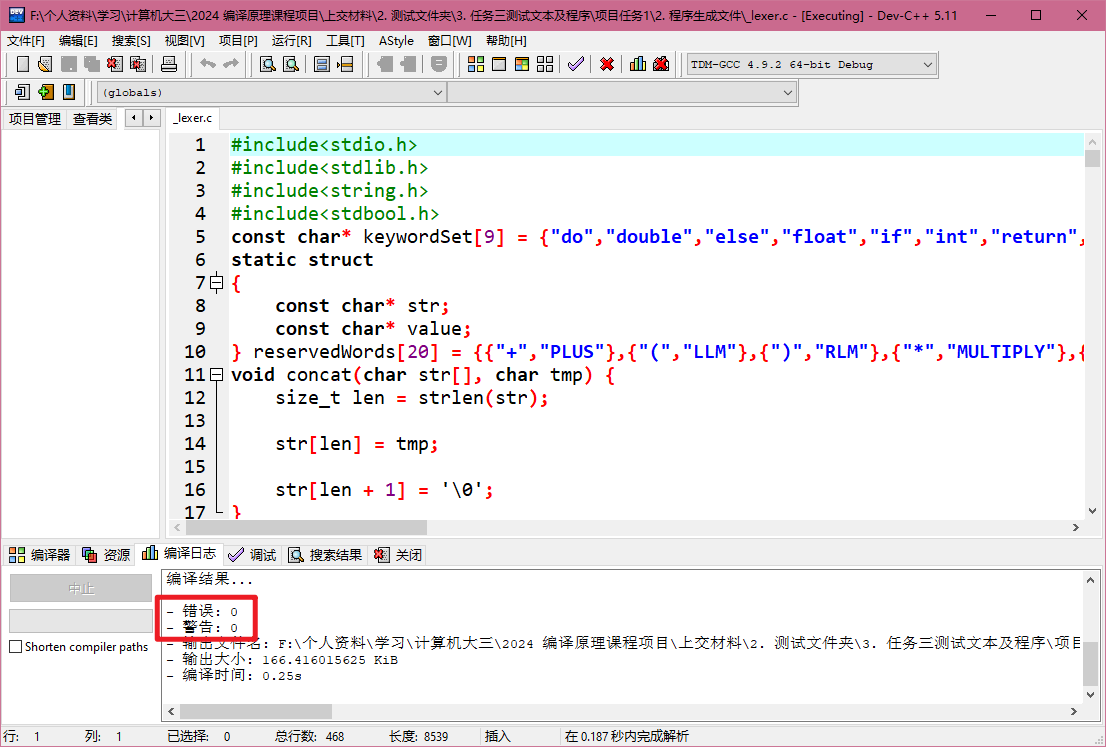


代码太长，具体可查看：

2. 测试文件夹\1. 任务一测试文本及程序\2. 程序生成文件中的\_lexer.c文件

**6、编译\_lexer.c文件并运行**

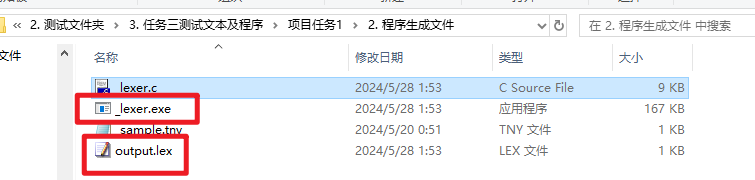
**注意：sample.tny必须和该程序放在同一个文件夹下。**



上图可知：编译成功

并成功生成lex文件：

**7、查看lex文件**





具体lex如下：

|  |
| --- |
| int:int  ID:x  SEMI:;  int:int  ID:fact  SEMI:;  double:double  ID:y  SEMI:;  int:int  ID:arr  LMM:[  NUMBER:100  RMM:]  SEMI:;  void:void  ID:test  LLM:(  int:int  ID:a  DOU:,  int:int  ID:b  RLM:)  LBM:{  ID:a  ASSIGN:=  NUMBER:1  SEMI:;  ID:b  ASSIGN:=  NUMBER:2  SEMI:;  return:return  SEMI:;  RBM:}  void:void  ID:main  LLM:(  void:void  RLM:)  LBM:{  ID:x  ASSIGN:=  NUMBER:2  SEMI:;  ID:y  ASSIGN:=  NUMBER:3  SEMI:;  if:if  LLM:(  ID:x  RT:>  NUMBER:1  RLM:)  do:do  ID:fact  ASSIGN:=  ID:fact  MULTIPLY:\*  ID:x  SEMI:;  while:while  LLM:(  ID:fact  RTEQ:>=  NUMBER:1  RLM:)  SEMI:;  else:else  ID:x  ASSIGN:=  NUMBER:1  SEMI:;  return:return  SEMI:;  RBM:}  EOF:EOF |

对照minic源程序，可知解析完全正确。

**测试结果**

任务三-项目一的测试**完全通过**

**二、项目任务二的测试**

**1、准备工作**

**minic的文法：**

|  |
| --- |
| program | definition-list | definition | variable-definition | function-definition | type-indicator | parameters | compound-stmt | parameter-list | parameter | local-definitions | statement-list | statement | expression-stmt | condition-stmt | dowhile-stmt | return-stmt | expression | simple-expression | variable | additive-expression | relop | term | addop | mulop | factor | call | arguments | argument-list  ID | SEMI | LMM | RMM | int | float | double | void | LLM | RLM | if | else | do | while | return | LTEQ | LT | RT | RTEQ | EQ | NE | PLUS | MINUS | MULTIPLY | DIVIDE | MOD | NUMBER | DOU | LBM | RBM | ASSIGN  program -> definition-list  definition-list -> definition-list definition  definition-list -> definition  definition -> variable-definition  definition -> function-definition  variable-definition -> type-indicator ID SEMI  variable-definition -> type-indicator ID LMM NUMBER RMM SEMI  type-indicator -> int  type-indicator -> float  type-indicator -> double  type-indicator -> void  function-definition -> type-indicator ID LLM parameters RLM compound-stmt  parameters -> parameter-list  parameters -> void  parameter-list -> parameter-list DOU parameter  parameter-list -> parameter  parameter -> type-indicator ID  parameter -> type-indicator ID LMM RMM  compound-stmt -> LBM local-definitions statement-list RBM  local-definitions -> local-definitions variable-definition  local-definitions -> @  statement-list -> statement-list statement  statement-list -> @  statement -> expression-stmt  statement -> compound-stmt  statement -> condition-stmt  statement -> dowhile-stmt  statement -> return-stmt  expression-stmt -> expression SEMI  expression-stmt -> SEMI  condition-stmt -> if LLM expression RLM statement  condition-stmt -> if LLM expression RLM statement else statement  dowhile-stmt -> do statement while LLM expression RLM SEMI  return-stmt -> return SEMI  return-stmt -> return expression SEMI  expression -> variable ASSIGN expression  expression -> simple-expression  variable -> ID  variable -> ID LMM expression RMM  simple-expression -> additive-expression relop additive-expression  simple-expression -> additive-expression  relop -> LTEQ  relop -> LT  relop -> RT  relop -> RTEQ  relop -> EQ  relop -> NE  additive-expression -> additive-expression addop term  additive-expression -> term  addop -> PLUS  addop -> MINUS  term -> term mulop factor  term -> factor  mulop -> MULTIPLY  mulop -> DIVIDE  mulop -> MOD  factor -> LLM expression RLM  factor -> variable  factor -> call  factor -> NUMBER  call -> ID LLM arguments RLM  arguments -> argument-list  arguments -> @  argument-list -> argument-list DOU expression  argument-list -> expression |

**语义函数：**

|  |
| --- |
| 0  0 1  0  0  0  1 0 -1  1 0 -1 2 -1 -1  0  0  0  0  1 0 -1 2 -1 3  0  0  0 -1 1  0  1 0  1 0 -1 -1  -1 0 1 -1  0 1  -1  0 1  -1  0  0  0  0  0  0 -1  -1  0 -1 1 -1 2  0 -1 1 -1 2 -1 3  0 1 -1 -1 2 -1 -1  0 -1  0 1 -1  1 0 2  0  0  0 -1 1 -1  1 0 2  0  0  0  0  0  0  0  0 1 2  0  0  0  0 1 2  0  0  0  0  -1 0 -1  0  0  0  0 -1 1 -1  0  -1  0 -1 1  0 |

**输入文法：**



**2、求解first集合**



项目过多，在此不一一展示。

**3、求解follow集合**



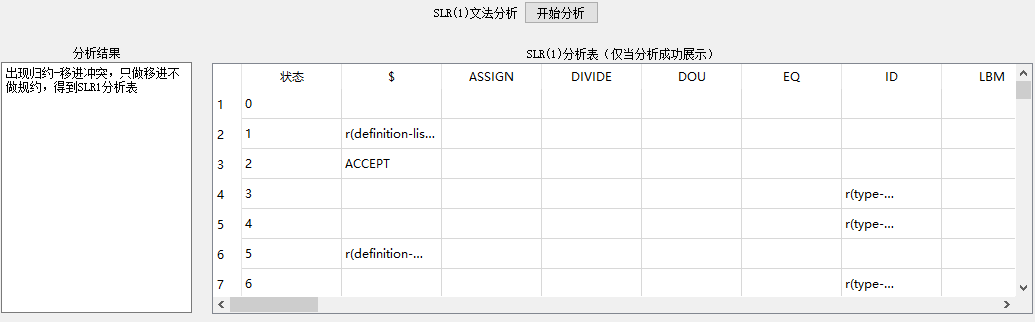
项目过多，在此不一一展示。

**4、LR0**



**项目太多，未能展示完全**

**5、SLR1表**

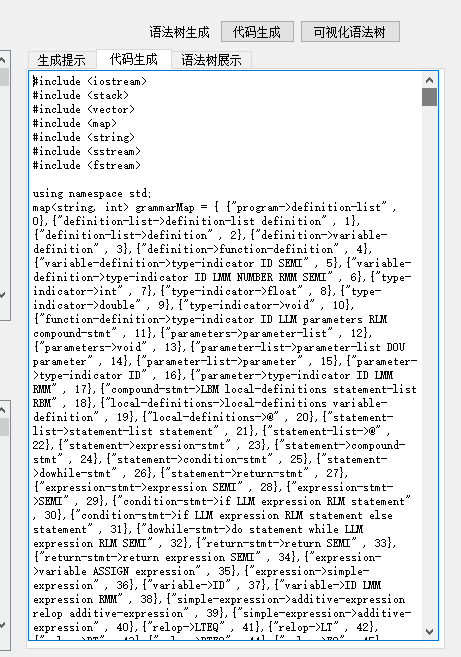


**项目太多，未能展示完全**

**出现移进归约冲突，我们做只移进不规约的处理**

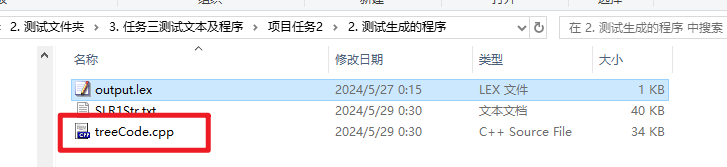
**6、语法树代码生成**

**注意：lex所在路径即为代码生成路径**



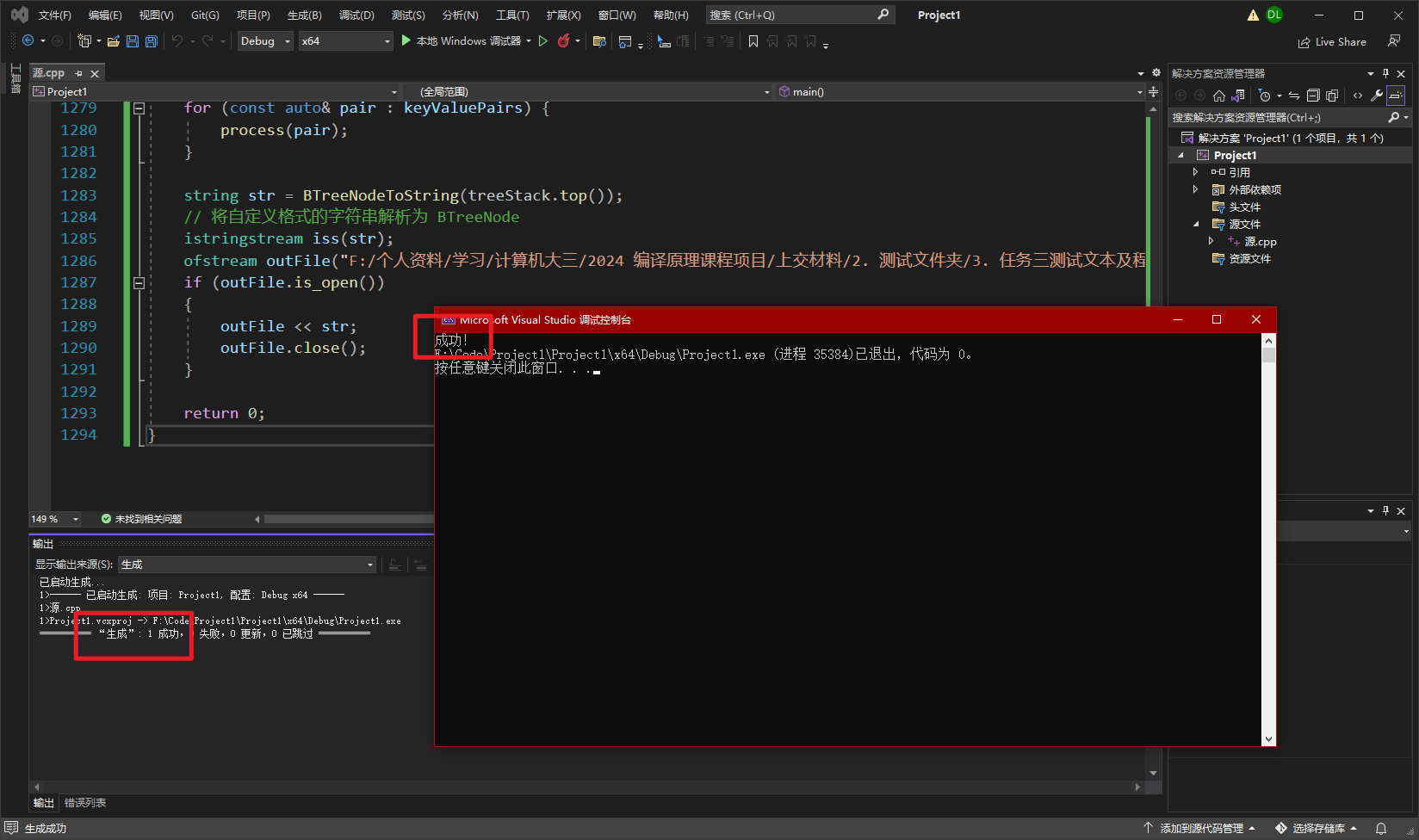
**7、查看语法树生成代码文件**

**PS:请用C++11以上进行编译**

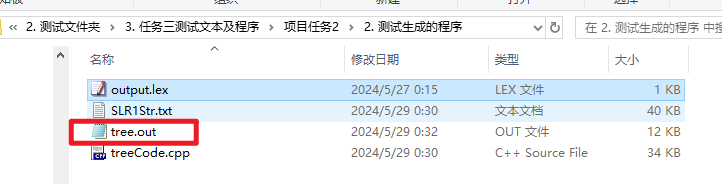


**具体语法树代码太长，在此不进行展示，请打开treeCode.cpp查看**

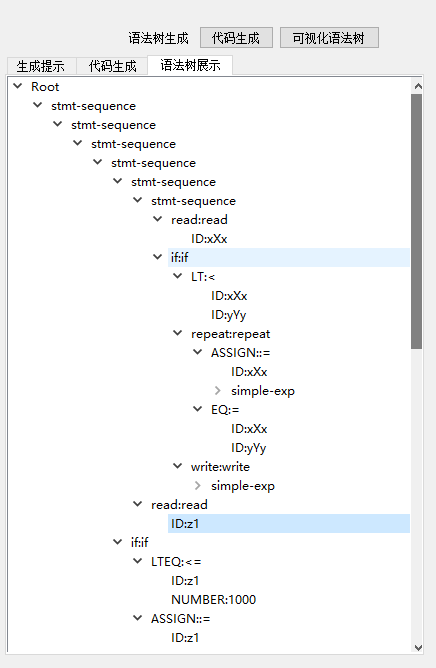
编译运行：



得到语法树：



**8、可视化语法树**



**测试结果**

任务三-项目二的测试**完全通过**