

**院 系：计 算 机 学 院**

**实验课程：编译原理**

**实验项目：TINY扩充语言的语法分析**

**指导老师：黄煜廉**

**专 业：计算机科学与技术**

**班 级：2018级 3 班**

**学 生：**

**学 号：**

**华南师范大学教务处**

**TINY扩充语言的语法分析**

**一．实验目的**

设计一个应用软件，可以对扩充的TINY语言进行语法分析，并可以生成语法树

1. **实验内容**
2. 扩充的语法规则有：实现 while、dowhile、for语句、+= 减法赋值运算符号（类似于C语言的+=）、求余%、乘方^、<=(小于等于)、<>(不等于)运算符号，具体文法规则自行构造，即为：
3. 运算符号的扩充：+= 减法赋值运算符号（类似于C语言的+=）、求余%、乘方^、<=(小于等于)、<>(不等于)运算符号。
4. While-stmt --> while(exp) stmt-sequence endwhile
5. Dowhile-stmt-->do stmt-sequence while(exp);
6. for-stmt-->for identifier:=simple-exp to simple-exp do stmt-sequence enddo 步长递增1
7. for-stmt-->for identifier:=simple-exp downto simple-exp do stmt-sequence enddo 步长递减1
8. 把TINY语言原有的if语句书写格式

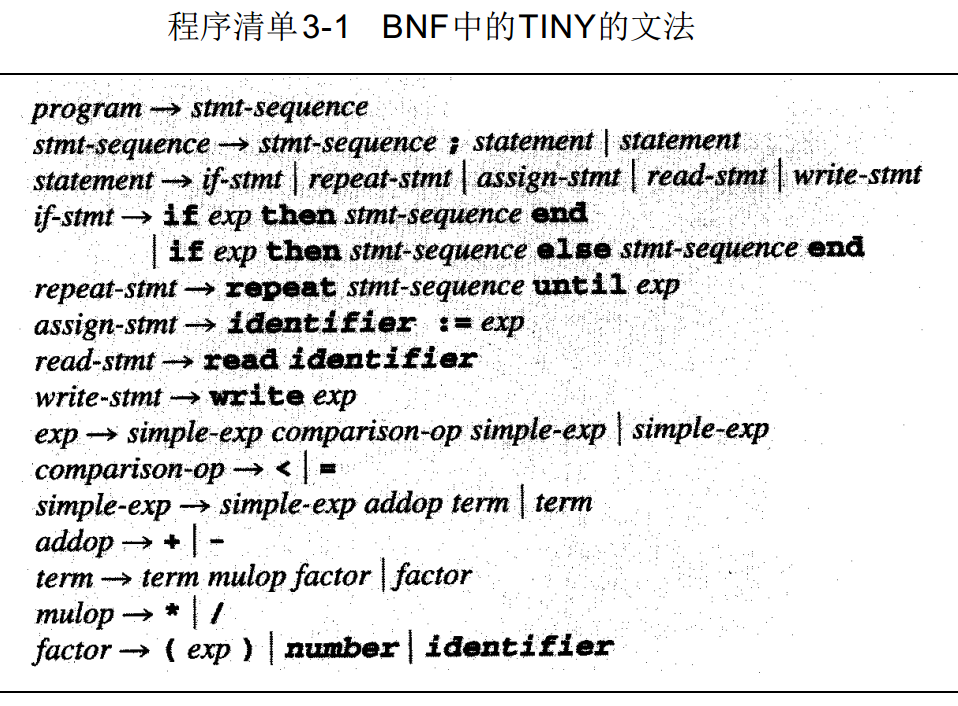
if\_stmt-->if exp then stmt-sequence end | if exp then stmt-sequence else stmt-sequence end

改写为：

if\_stmt-->if(exp) stmt-sequence else stmt-sequence | if(exp) stmt-sequence

**三．设计思路**

1. 老师已在作业中给出TINY语言的文法规则：

****

1. 按照题目要求对原来的语法规则进行修改，即可得到经过修改后的文法规则如下：
2. program→stmt-sequence
3. stmt-sequence → stmt-sequence ; statement | statement
4. statement → if-stmt | repeat-stmt I assign-stmt | read-stmt | write-stmt |while-stmt | for-stmt Dowhile-stmt
5. if\_stmt → if(exp) stmt-sequence else stmt-sequence | if(exp) stmt-sequence
6. repeat-stmt → repeat stmt-sequence until exp
7. assign-stmt→identifier := exp
8. read-stmt-→read identifier
9. write-stmt →write exp
10. exp→simple-exp comparison-op simple-exp | simple-exp
11. comparison-op → < | = | <= | <>
12. simple-exp → simple-exp addop term | term
13. addop→+| - | +=
14. term → term mulop factor| factor
15. mulop→\*| / |% | ^
16. factor→( exp)| number | identifier
17. Dowhile-stmt → do stmt-sequence while(exp)
18. While-stmt → while(exp) stmt-sequence endwhile
19. for-stmt → for identifier:=simple-exp to | downto fornext
20. fornext → simple-exp do stmt-sequence enddo
21. 由上课时所讲的知识可将消去左递归，其转变得到EBNF如下：
22. program→stmt-sequence
23. stmt-sequence → statement { ;statement }
24. statement → if-stmt | repeat-stmt I assign-stmt | read-stmt | write-stmt |while-stmt | for-stmt | Dowhile-stmt
25. if\_stmt→if(exp) stmt-sequence [else stmt-sequence]
26. repeat-stmt → repeat stmt-sequence until exp
27. assign-stmt→identifier := exp
28. read-stmt→read identifier
29. write-stmt →write exp
30. Dowhile-stmt → do stmt-sequence while(exp)
31. for-stmt → for identifier:=simple-exp to | downto fornext
32. fornext → simple-exp do stmt-sequence enddo
33. while-stmt → while(exp) stmt-sequence endwhile
34. exp→simple-exp [ comparison-op simple-exp ]
35. comparison-op → < | = | <= | <>
36. simple-exp → term {addop term}
37. addop→+| - | +=
38. term → factor {mulop factor}
39. mulop→\*| / |% | ^
40. factor→( exp)| number | identifier
41. 再由课堂知识可以得到各个非终结符的first集合如下：

(分别用NUM和ID来代指number和identifier)

1. first(factor)={(,NUM,ID}
2. first(mulop)={\*,/,%,^}
3. first(term)=first(factor)={(,NUM,ID}
4. first(addop)={+,-,+=}
5. first(simple-exp)= first(term)= first(factor)={(,NUM,ID}
6. first(comparison-op)={<,=,<=,<>}
7. first(exp)=first(simple-exp)={(,NUM,ID}
8. first(while-stmt)={while}
9. first(fornext)=first(simple-exp}={,NUM,ID}
10. first(for-stmt)={for}
11. first(dowhile-stmt)={do}
12. first (write-stmt)={write}
13. first (read-stmt)={read}
14. first (assign-stmt)={ID}
15. first(repeat-stmt)={repeat}
16. first(if-stmt)={if}
17. first(statement)={if,reapeat,ID,read,write,do,for ,while}
18. first(stmt-sequence)={if,reapeat,ID,read,write,do,for, while}

**四．程序源代码**

|  |
| --- |
| analys.h |
| #ifndef ANALYSIS\_H  #define ANALYSIS\_H  #include "iostream"  #include "stdio.h"  #include "string.h"  #include "map"  #include "vector"  #include "ctype.h"  #include "mainwindow.h"  #include "ui\_mainwindow.h"  using namespace std;  typedef enum{  //结束  ENDFILE, ERROR\_SYMBOL,  //文法符号(最后一个为；)  IF, THEN, ELSE, END, REPEAT, UNTIL, READ, WRITE, WHILE, DO, ENDWHILE, FOR, TO, DOWNTO, ENDDO, SPLIT,  //特殊组合符号  ID, NUM,  //运算符号(:=，（，），\*，+，+=，^，%，<，=，<=，<>，-，/)  ASSIGN, LEFT, RIGHT, MULT, PLUS, PLUSEQUAL, TIMES, COMPLE, LESS, EQUAL, LESSEQUAL, UNEQUAL, REDUCE, DIVIDE,  } TokenTypeInProject;  class **Gram**{  public:  **Gram**(){tokenno=0; lineno=0; innotes=false; tree="statements{ ";} //初始化  int **Mid**(); //内部主函数  void **gettext**(QString QS); //获得文本并分列  vector<string>**getERROR**(){return ERROR\_MES;} //得到错误信息  string **gettree**(){return tree;}  private:  bool innotes;  int tokenno;  int lineno;  string temp;  string tree;  //string tree;  vector<string> text;  vector<string> ERROR\_MES;  TokenTypeInProject token;  TokenTypeInProject TEMP;  string nowtext;  //对输入的文本进行处理用  TokenTypeInProject **getnexttoken**(); //获得下一个符号  string **getnextline**(); //获得下一列文本  //对文本语法探测用  TokenTypeInProject **program**();  TokenTypeInProject **stmt\_sequence**();  TokenTypeInProject **statement**();  TokenTypeInProject **if\_stmt**();  TokenTypeInProject **repeat\_stmt**();  TokenTypeInProject **assign\_stmt**();  TokenTypeInProject **read\_stmt**();  TokenTypeInProject **write\_stmt**();  TokenTypeInProject **dowhile\_stmt**();  TokenTypeInProject **for\_stmt**();  TokenTypeInProject **while\_stmt**();  TokenTypeInProject **exp**();  TokenTypeInProject **com\_op**();  TokenTypeInProject **simple\_exp**();  TokenTypeInProject **addop**();  TokenTypeInProject **term**();  TokenTypeInProject **mulop**();  TokenTypeInProject **factor**();  TokenTypeInProject **Maket**(TokenTypeInProject expect); //匹配  void **Error**(string s); //错误函数  };  #endif // ANALYSIS\_H |

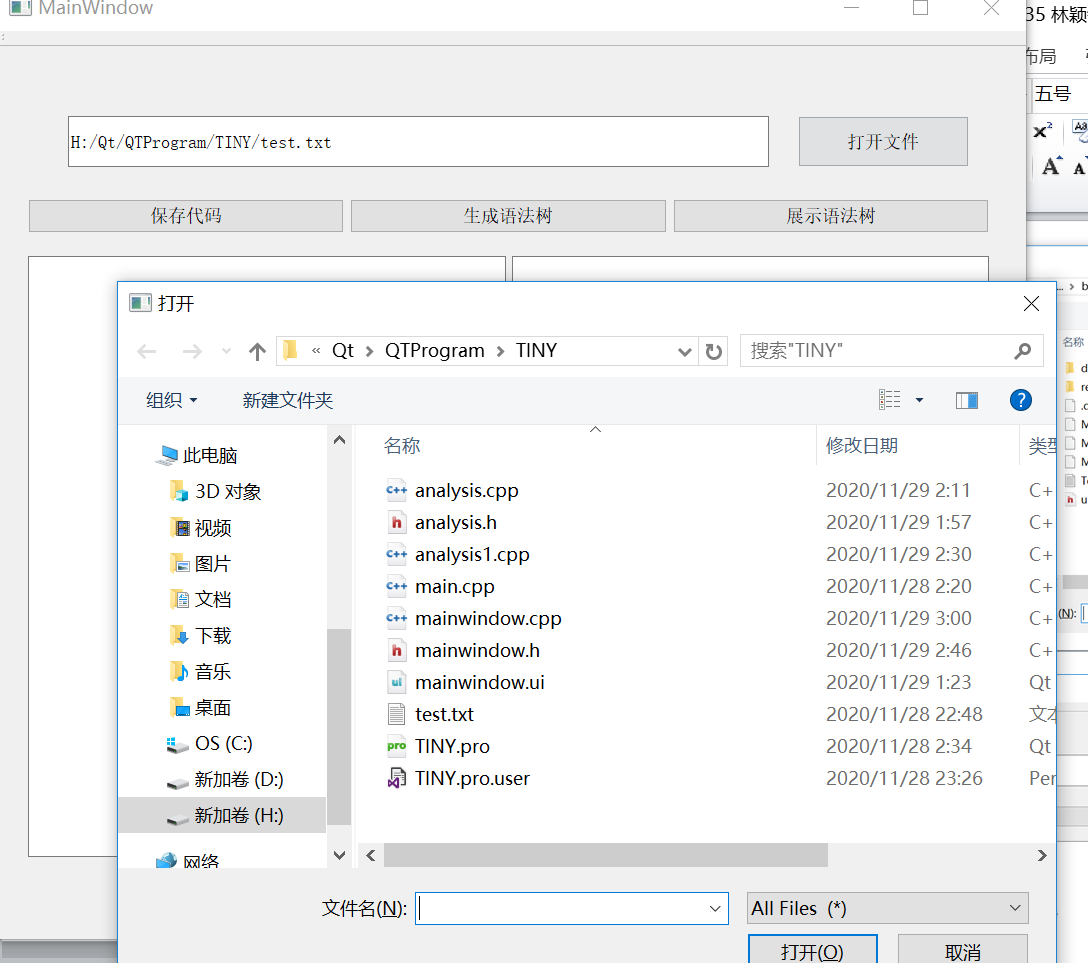
|  |
| --- |
| analys.cpp |
| #include "analysis.h"  //内部主函数  int Gram::**Mid**() {  do{  nowtext=text[lineno];  TEMP=getnexttoken(); if(TEMP==ENDFILE) break;  TEMP=program(); if(TEMP==ENDFILE) break;  nowtext=getnextline(); if(nowtext=="EOP") break;  }while(lineno<text.size()-1);  tree+="}";  return 0;  }  //获得下一个字符  TokenTypeInProject Gram::**getnexttoken**(){  temp="";  while(innotes==true){  for(int t=tokenno; t<nowtext.length(); t++){  if(nowtext[t]=='}'){ //注释结束  innotes=false;  }  }  nowtext=getnextline();  }  while(tokenno>=nowtext.length()) nowtext=getnextline();  int i=tokenno;  while(nowtext[i]==' ') i++;//去除前面空格  if(!innotes){  //拼合  //为字母情况  if(isalpha(nowtext[i])){  for(; i<nowtext.length(); i++){  if(isalpha(nowtext[i]))  temp += nowtext[i];  else break;  }  }  //为数字情况  else if(isdigit(nowtext[i])){  for(; i<nowtext.length(); i++){  if(isdigit(nowtext[i]))  temp += nowtext[i];  else break;  }  }  //为注释情况  else if(nowtext[i]=='{'){  temp+=nowtext[i];  }  //else if(nowtext[i]==';'){  // temp+=nowtext[i];  //}  //为其他符号情况  else if(nowtext[i]!=' '){  for(; i<nowtext.length(); i++){  if(!isdigit(nowtext[i]) && !isalpha(nowtext[i]) && nowtext[i]!=' ' && nowtext[i]!='{')  {temp += nowtext[i];}  else break;  }  }  tokenno=i;  //筛选  if(isalpha(temp[0])){ //首字符为字母  if(temp=="if") {token=IF;}  else if(temp=="then") {token=THEN;}  else if(temp=="else") {token=ELSE;}  else if(temp=="end") token=END;  else if(temp=="repeat") {token=REPEAT;}  else if(temp=="until") {token=UNTIL;}  else if(temp=="read") {token=READ;}  else if(temp=="write") {token=WRITE;}  else if(temp=="while") {token=WHILE;}  else if(temp=="do") {token=DO;}  else if(temp=="endwhile") {token=ENDWHILE;}  else if(temp=="for") {token=FOR;}  else if(temp=="to") {token=TO;}  else if(temp=="downto") {token=DOWNTO;}  else if(temp=="enddo") token=ENDDO;  else if(temp=="EOP") token=ENDFILE;  else { token=ID; tree=tree+"ID:"+temp+", ";}  }  else if(isdigit(temp[0])){  token=NUM;tree=tree+"NUM:"+temp+", ";  }  else{  if(temp==":=") {token=ASSIGN;}  else if(temp=="(") {token=LEFT;}  else if(temp==")") {token=RIGHT;}  else if(temp=="\*") {token=MULT; tree=tree+"op:\*,";}  else if(temp=="+") {token=PLUS; tree=tree+"op:+,";}  else if(temp=="+=") {token=PLUSEQUAL; tree=tree+"op:+=,";}  else if(temp=="^") {token=TIMES; tree=tree+"op:^,";}  else if(temp=="%") {token=COMPLE; tree=tree+"op:%,";}  else if(temp=="<") {token=LESS; tree=tree+"op:<,";}  else if(temp=="=") {token=EQUAL; tree=tree+"op:=,";}  else if(temp=="<=") {token=LESSEQUAL; tree=tree+"op:<=,";}  else if(temp=="<>") {token=UNEQUAL; tree=tree+"op:<=,";}  else if(temp==";") {token=SPLIT;}  else if(temp=="-") {token=REDUCE; tree=tree+"op:-,";}  else if(temp=="/") {token=DIVIDE; tree=tree+"op:/,";}  else if(temp[0]=='{'){ //为注释情况  innotes=!innotes;  token=getnexttoken();  }  }  }  return token;  }  //得到文本并进行拆分  void Gram::**gettext**(QString QS){  string s=QS.toStdString();  char ch='\n'; int pos;  s.push\_back(ch);  int start = 0,end = 0,size = s.size();  for(int i = 0;i < size;i++){  if(s[i]==ch){  end = i;  string temp = s.substr(start,end-start);  start = i+1;  text.push\_back(temp);  }  }  text.push\_back("EOP");  }  //得到下一行  string Gram::**getnextline**(){  lineno=lineno+1; tokenno=0;  //if(text[lineno]=="EOP") system("pause");  return text[lineno];  } |

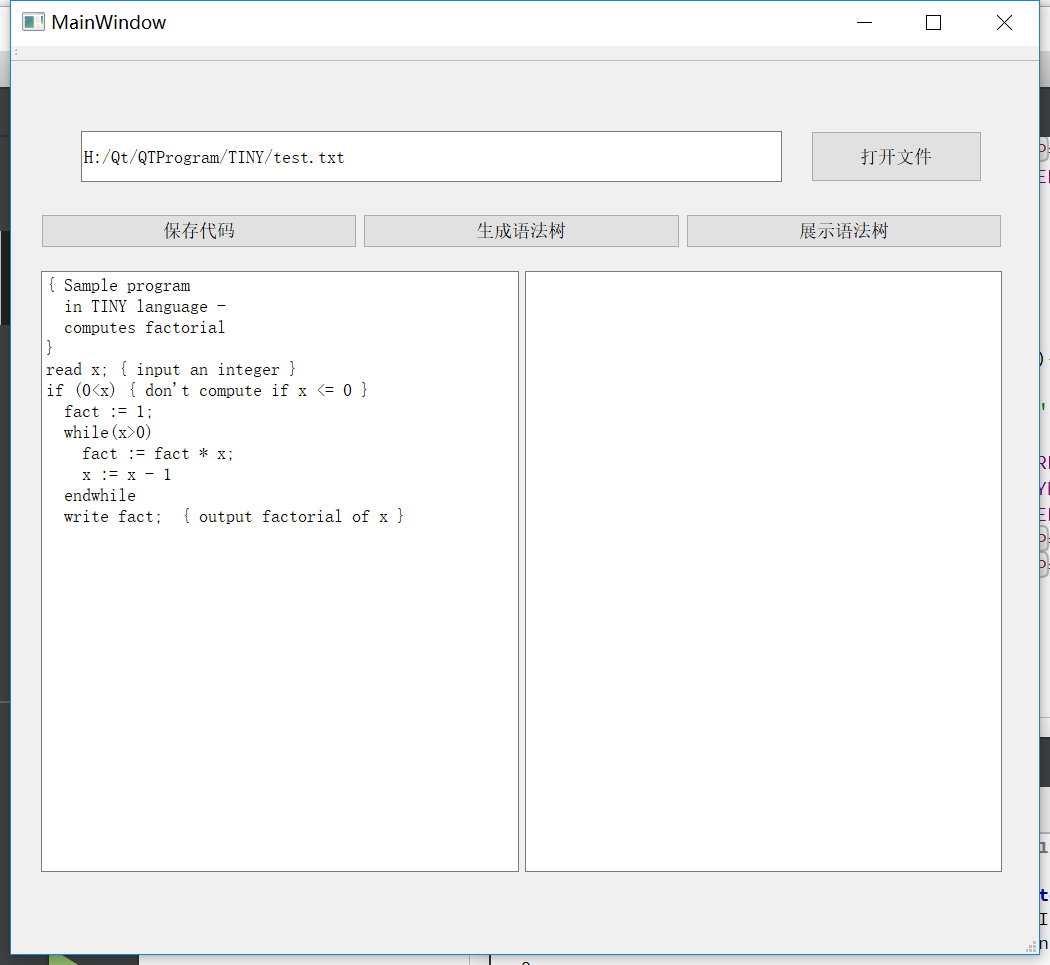
|  |
| --- |
| analysis1.cpp |
| #include "analysis.h"  //匹配  TokenTypeInProject Gram::**Maket**(TokenTypeInProject expect){  if(token!=expect){ //不匹配  return ERROR\_SYMBOL;  }  else{  token=getnexttoken();  //if(token==ENDFILE) cout<<"endfile"<<endl;  return token;  }  }  //错误  void Gram::**Error**(string s){  string error\_msg="Syntax error at line ";  error\_msg += to\_string(lineno+1);  error\_msg += ": ";  error\_msg += s;  ERROR\_MES.push\_back(error\_msg);  getnextline();  }  TokenTypeInProject Gram::**program**(){  TEMP=stmt\_sequence();  if(TEMP==ERROR\_SYMBOL) {Error(nowtext);}  return TEMP;  }  TokenTypeInProject Gram::**stmt\_sequence**(){  if(token==IF || token==REPEAT || token==ID || token==READ ||token== WRITE || token==DO || token==FOR ||token==WHILE){  TEMP=statement(); if(TEMP==ERROR\_SYMBOL) return ERROR\_SYMBOL;  while(token==SPLIT){  TEMP=Maket(SPLIT); if(TEMP==ENDFILE) return ENDFILE;  TEMP=statement(); if(TEMP==ERROR\_SYMBOL) {return ERROR\_SYMBOL;}  }  }  else {TEMP=ERROR\_SYMBOL;}  return TEMP;  }  TokenTypeInProject Gram::**statement**(){  switch (token){  case IF: {TEMP=if\_stmt();break;}  case REPEAT: {TEMP=repeat\_stmt(); break;}  case ID: {TEMP=assign\_stmt();break;}  case READ: {TEMP=read\_stmt();break;}  case WRITE: {TEMP=write\_stmt();break;}  case WHILE: {TEMP=while\_stmt();break;}  case FOR: {TEMP=for\_stmt();break;}  case DO: {TEMP=dowhile\_stmt();break;}  default:{  TEMP=ERROR\_SYMBOL;  break;  }  }  return TEMP;  }  TokenTypeInProject Gram::**if\_stmt**(){  if(token==IF){  tree=tree+"if-stmt{if:";  TEMP=Maket(IF);  TEMP=Maket(LEFT); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  TEMP=exp(); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;} else{tree+=";if\_body:";}  TEMP=Maket(RIGHT); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  TEMP=stmt\_sequence(); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  if(token==ELSE){  TEMP=Maket(ELSE); tree+="; else\_body:";  TEMP=stmt\_sequence();  if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  }  tree+="}";  }  else {TEMP=ERROR\_SYMBOL;}  return TEMP;  }  TokenTypeInProject Gram::**repeat\_stmt**(){  if(token==REPEAT){  tree=tree+"repeat-stmt{repeat\_body:";  TEMP=Maket(REPEAT);  TEMP=stmt\_sequence(); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;} else{tree+=";repeat:";}  TEMP=Maket(UNTIL); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  TEMP=exp(); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  tree+="}";  }  else {TEMP=ERROR\_SYMBOL;}  return TEMP;  }  TokenTypeInProject Gram::**assign\_stmt**(){  if(token==ID){  tree=tree+"assign{ ID:"+temp+",";  TEMP=Maket(ID);  TEMP=Maket(ASSIGN); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  TEMP=exp(); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  tree+="}";  }  else {TEMP=ERROR\_SYMBOL;}  return TEMP;  }  TokenTypeInProject Gram::**read\_stmt**(){  if(token==READ){  tree=tree+"read-stmt{ ";  TEMP=Maket(READ); tree=tree+"op:read,";  TEMP=Maket(ID); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  tree+="}";  }  else {TEMP=ERROR\_SYMBOL;}  return TEMP;  }  TokenTypeInProject Gram::**write\_stmt**(){  if(token==WRITE){  tree=tree+"write-stmt{ ";  TEMP=Maket(WRITE); tree=tree+"op:write,";  TEMP=exp(); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  tree+="}";  }  else {TEMP=ERROR\_SYMBOL;}  return TEMP;  }  TokenTypeInProject Gram::**dowhile\_stmt**(){  if(token==DO){  tree=tree+"dowhile-stmt{dowhile\_body: ";  TEMP=Maket(DO);  TEMP=stmt\_sequence(); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  TEMP=Maket(WHILE); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}else{tree+="; while:";}  TEMP=Maket(LEFT); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  TEMP=exp(); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  TEMP=Maket(RIGHT); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  tree+="}";  }  else {TEMP=ERROR\_SYMBOL;}  return TEMP;  }  TokenTypeInProject Gram::**for\_stmt**(){  if(token==FOR){  tree=tree+"for-stmt{ ";  TEMP=Maket(FOR);  TEMP=Maket(ID); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  TEMP=Maket(ASSIGN); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  TEMP=simple\_exp(); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  switch (token){  case TO:{TEMP=Maket(TO);break;}  case DOWNTO: {TEMP=Maket(DOWNTO);break;}  default:{TEMP=ERROR\_SYMBOL; {tree+="}"; return ERROR\_SYMBOL;}}  }  TEMP=simple\_exp(); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;} else{tree+=";for\_body:";}  TEMP=Maket(DO); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  TEMP=stmt\_sequence(); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  TEMP=Maket(ENDDO); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  tree+="}";  }  else {TEMP=ERROR\_SYMBOL;}  return TEMP;  }  TokenTypeInProject Gram::**while\_stmt**(){  if(token==WHILE){  tree=tree+"while-stmt{while:";  TEMP=Maket(WHILE);  TEMP=Maket(LEFT); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  TEMP=exp(); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;} else{tree+=";while\_body:";}  TEMP=Maket(RIGHT); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  TEMP=stmt\_sequence(); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  TEMP=Maket(ENDWHILE); if(TEMP==ERROR\_SYMBOL) {tree+="}"; return ERROR\_SYMBOL;}  tree+="}";  }  else {TEMP=ERROR\_SYMBOL;}  return TEMP;  }  TokenTypeInProject Gram::**exp**(){  if(token==LEFT || token==NUM || token==ID){  TEMP=simple\_exp(); if(TEMP==ERROR\_SYMBOL) return ERROR\_SYMBOL;  if(token==LESS || token==EQUAL || token==LESSEQUAL || token==UNEQUAL){  TEMP=com\_op(); if(TEMP==ERROR\_SYMBOL) return ERROR\_SYMBOL;  TEMP=simple\_exp(); if(TEMP==ERROR\_SYMBOL) return ERROR\_SYMBOL;  }  }  else {TEMP=ERROR\_SYMBOL;}  return TEMP;  }  TokenTypeInProject Gram::**com\_op**(){  switch(token){  case LESS:{TEMP=Maket(LESS);break;}  case EQUAL:{TEMP=Maket(EQUAL);break;}  case LESSEQUAL:{TEMP=Maket(LESSEQUAL);break;}  case UNEQUAL:{TEMP=Maket(UNEQUAL);break;}  default:{TEMP=ERROR\_SYMBOL;break;}  }  return TEMP;  }  TokenTypeInProject Gram::**simple\_exp**(){  if(token==LEFT || token==NUM || token==ID){  TEMP=term(); if(TEMP==ERROR\_SYMBOL) return ERROR\_SYMBOL;  while(token==PLUS || token==REDUCE || token==PLUSEQUAL){  TEMP=addop(); if(TEMP==ERROR\_SYMBOL) return ERROR\_SYMBOL;  TEMP=term(); if(TEMP==ERROR\_SYMBOL) return ERROR\_SYMBOL;  }  }  else {TEMP=ERROR\_SYMBOL;}  return TEMP;  }  TokenTypeInProject Gram::**addop**(){  switch(token){  case PLUS:{TEMP=Maket(PLUS);break;}  case REDUCE:{TEMP=Maket(REDUCE);break;}  case PLUSEQUAL:{TEMP=Maket(PLUSEQUAL);break;}  default:{TEMP=ERROR\_SYMBOL;break;}  }  return TEMP;  }  TokenTypeInProject Gram::**term**(){  if(token==LEFT || token==NUM || token==ID){  TEMP=factor(); if(TEMP==ERROR\_SYMBOL) return ERROR\_SYMBOL;  while(token==MULT || token==DIVIDE || token==TIMES || token==COMPLE){  TEMP=mulop(); if(TEMP==ERROR\_SYMBOL) return ERROR\_SYMBOL;  TEMP=factor(); if(TEMP==ERROR\_SYMBOL) return ERROR\_SYMBOL;  }  }  else {TEMP=ERROR\_SYMBOL;}  return TEMP;  }  TokenTypeInProject Gram::**mulop**(){  switch(token){  case MULT:{TEMP=Maket(MULT);break;}  case DIVIDE:{TEMP=Maket(DIVIDE);break;}  case TIMES:{TEMP=Maket(TIMES);break;}  case COMPLE:{TEMP=Maket(COMPLE);break;}  default:{TEMP=ERROR\_SYMBOL;break;}  }  return TEMP;  }  TokenTypeInProject Gram::**factor**(){  if(token==LEFT){  TEMP=Maket(LEFT);  TEMP=exp(); if(TEMP==ERROR\_SYMBOL) return ERROR\_SYMBOL;  TEMP=Maket(RIGHT); if(TEMP==ERROR\_SYMBOL) return ERROR\_SYMBOL;  }  else if(token==NUM){  TEMP=Maket(NUM);  }  else if(token==ID){  TEMP=Maket(ID);  }  else {TEMP=ERROR\_SYMBOL;}  return TEMP;  } |

**五．实验结果**

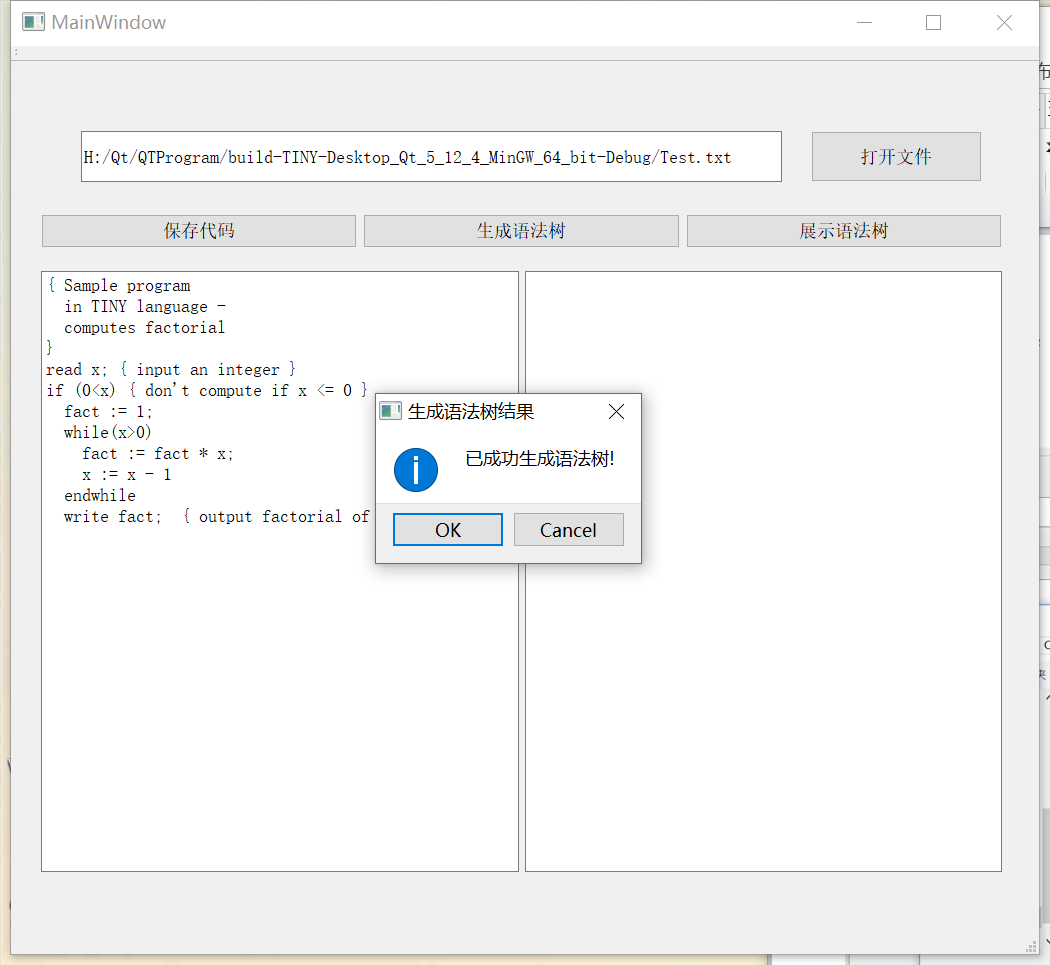
运行程序，截图如下：

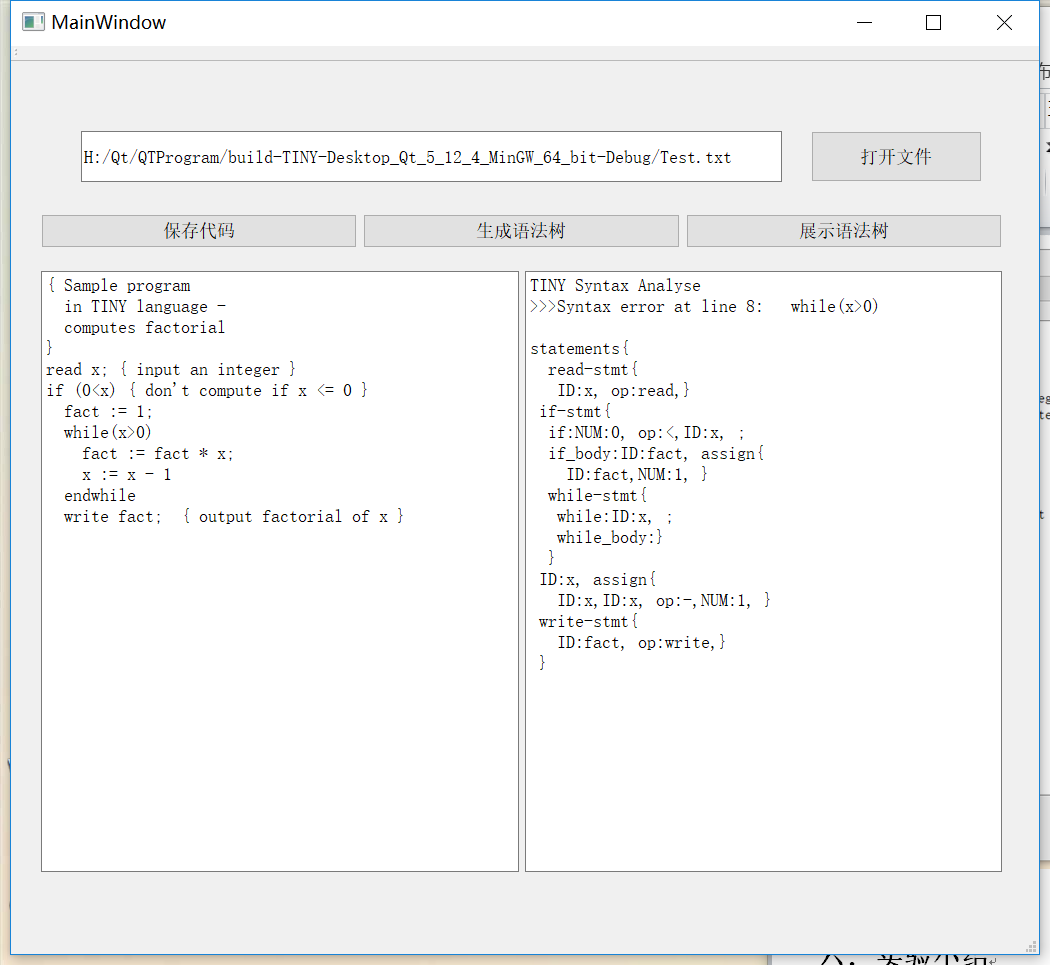
1. 打开文件



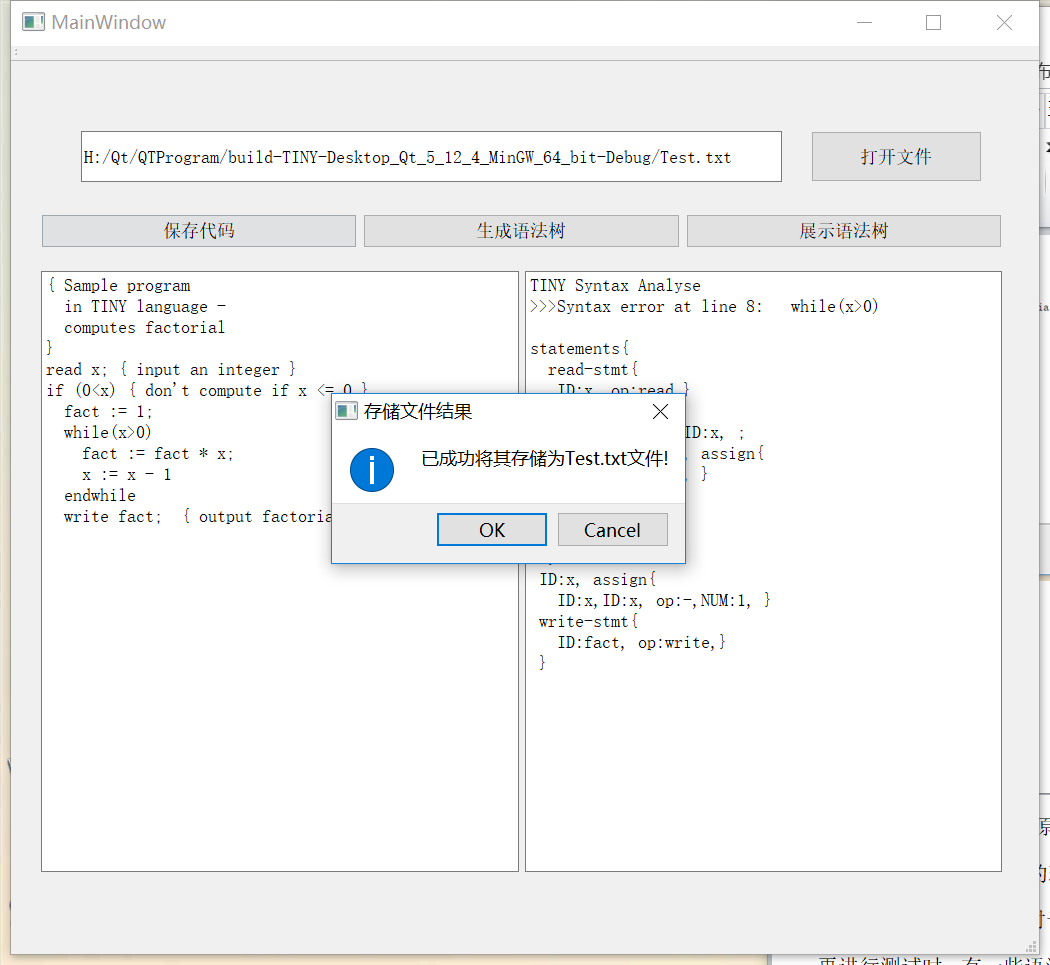


1. 生成和查看语法树





1. 保存代码



**六．实验小结**

本次实验通过对编译原理中TINY语法分析的实现，更加深刻地了解了语法分析的相关知识，对于递归下降法的理解更进一步。但经过测试，本程序对于TINY的扩充语法可以很好的进行分析，并且可以提示错误，但目前错误显示仅仅能显示错误所在行数。

1. **实验小结**

参考资料：

1. :黄煜廉老师的ppt