**《编译原理》实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | 2019级计算机科学与技术01班 | | **姓名** | 谢双骏 |
| **实验题目** | 语法分析程序的设计与实现 | | | | |
| **实验时间** | 2022.4.16 | | **实验地点** | DS3401 | |
| **实验成绩** |  | | **实验性质** | □验证性 □设计性 ■综合性 | |
| 教师评价：  □算法/实验过程正确；□源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； ☑语法、语义正确； □报告规范；  其他：  **评价教师签名：** | | | | | |
| 一、实验目的   1. 了解编译器中语法分析环节的作用和意义 2. 掌握语法分析的方法和步骤 3. 熟悉自底向上、自顶向下等各类语法分析方法；掌握语法树的构造方法 | | | | | |
| 二、实验项目内容  请根据给定的文法设计并实现语法分析程序，能基于上次作业的词法分析程序所识别出的单词，识别出各类语法成分。输入输出及处理要求如下：   1. 需按文法规则，用递归子程序法对文法中定义的所有种语法成分进行分析； 2. 为了方便进行自动评测，输入的被编译源文件统一命名为testfile.txt（注意不要写错文件名）；输出的结果文件统一命名为output.txt（注意不要写错文件名）；结果文件中包含如下两种信息： 3. 按词法分析识别单词的顺序，按行输出每个单词的信息（要求同词法分析作业，对于预读的情况不能输出）。 4. 在文法中高亮显示（见2020文法定义）的语法分析成分分析结束前，另起一行输出当前语法成分的名字，形如“<常量说明>”（注：未要求输出的语法成分仍需要分析）   【输入形式】testfile.txt中的符合文法要求的测试程序。  【输出形式】按如上要求将语法分析结果输出至output.txt中，中文字符的编码格式要求是UTF-8。  【特别提醒】   1. 本次作业只考核对正确程序的处理，但需要为今后可能出现的错误情况预留接口。   （2）当前要求的输出只是为了便于评测，完成编译器中无需出现这些信息，请设计为方便打开/关闭这些输出的方案。  【文法定义】请到“2020年文法定义(用于查看文法，不计入成绩）”作业中查看文法 | | | | | |
| 实验过程或算法  1. 完成内容   在本次实验中，我主要实现了一个语法分析器，代码核心部分在lexical\_analyse文件当中。  本次实验实现语法分析的思想主要是采用自顶向下的递归调用的方式去分析代码，并产生语法分析的结果，最后将平台上要求的分析结果输出到文件中。  在对输入语句分析的前提是对语句进行词法分析，主要依托实验一实现的词法分析程序，我们直接采用其分析出来的token序列即可。  对于错误处理部分，我们在本次试验中暂未考虑，只是在每个函数的编写时在每一个可能有问题的地方留下了else情况用来做错误处理的接口，以便在实验三时进行完善，由于本次实验的平台测试代码是完全正确的，所以没有影响。   1. 文法说明   主要的文法设计如下：  ＜加法运算符＞ ::= +｜-  ＜乘法运算符＞  ::= \*｜/  ＜关系运算符＞  ::=  <｜<=｜>｜>=｜!=｜==  ＜字母＞   ::= ＿｜a｜．．．｜z｜A｜．．．｜Z  ＜数字＞   ::= ０｜１｜．．．｜９  ＜字符＞    ::=  '＜加法运算符＞'｜'＜乘法运算符＞'｜'＜字母＞'｜'＜数字＞'  ＜字符串＞   ::=  "｛十进制编码为32,33,35-126的ASCII字符｝"   ＜程序＞    ::= ［＜常量说明＞］［＜变量说明＞］{＜有返回值函数定义＞|＜无返回值函数定义＞}＜主函数＞  ＜常量说明＞ ::=  const＜常量定义＞;{ const＜常量定义＞;}  ＜常量定义＞   ::=   int＜标识符＞＝＜整数＞{,＜标识符＞＝＜整数＞}  | char＜标识符＞＝＜字符＞{,＜标识符＞＝＜字符＞}  ＜无符号整数＞  ::= ＜数字＞｛＜数字＞｝  ＜整数＞        ::= ［＋｜－］＜无符号整数＞  ＜标识符＞    ::=  ＜字母＞｛＜字母＞｜＜数字＞｝  ＜声明头部＞   ::=  int＜标识符＞ |char＜标识符＞  ＜常量＞   ::=  ＜整数＞|＜字符＞  ＜变量说明＞  ::= ＜变量定义＞;{＜变量定义＞;}  ＜变量定义＞ ::= ＜变量定义无初始化＞|＜变量定义及初始化＞  ＜变量定义无初始化＞  ::= ＜类型标识符＞(＜标识符＞|＜标识符＞'['＜无符号整数＞']'|＜标识符＞'['＜无符号整数＞']''['＜无符号整数＞']'){,(＜标识符＞|＜标识符＞'['＜无符号整数＞']'|＜标识符＞'['＜无符号整数＞']''['＜无符号整数＞']' )}  ＜变量定义及初始化＞  ::= ＜类型标识符＞＜标识符＞=＜常量＞|＜类型标识符＞＜标识符＞'['＜无符号整数＞']'='{'＜常量＞{,＜常量＞}'}'|＜类型标识符＞＜标识符＞'['＜无符号整数＞']''['＜无符号整数＞']'='{''{'＜常量＞{,＜常量＞}'}'{, '{'＜常量＞{,＜常量＞}'}'}'}'  ＜类型标识符＞      ::=  int | char  ＜有返回值函数定义＞  ::=  ＜声明头部＞'('＜参数表＞')' '{'＜复合语句＞'}'  ＜无返回值函数定义＞  ::= void＜标识符＞'('＜参数表＞')''{'＜复合语句＞'}'  ＜复合语句＞   ::=  ［＜常量说明＞］［＜变量说明＞］＜语句列＞  ＜参数表＞    ::=  ＜类型标识符＞＜标识符＞{,＜类型标识符＞＜标识符＞}| ＜空＞  ＜主函数＞    ::= void main‘(’‘)’ ‘{’＜复合语句＞‘}’  ＜表达式＞    ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞}  ＜项＞     ::= ＜因子＞{＜乘法运算符＞＜因子＞}  ＜因子＞    ::= ＜标识符＞｜＜标识符＞'['＜表达式＞']'|＜标识符＞'['＜表达式＞']''['＜表达式＞']'|'('＜表达式＞')'｜＜整数＞|＜字符＞｜＜有返回值函数调用语＞  ＜语句＞    ::= ＜循环语句＞｜＜条件语句＞| ＜有返回值函数调用语句＞;  |＜无返回值函数调用语句＞;｜＜赋值语句＞;｜＜读语句＞;｜＜写语句＞;｜＜情况语句＞｜＜空＞;|＜返回语句＞; | '{'＜语句列＞'}'  ＜赋值语句＞   ::=  ＜标识符＞＝＜表达式＞|＜标识符＞'['＜表达式＞']'=＜表达式＞|＜标识符＞'['＜表达式＞']''['＜表达式＞']' =＜表达式＞  ＜条件语句＞  ::= if '('＜条件＞')'＜语句＞［else＜语句＞］  ＜条件＞    ::=  ＜表达式＞＜关系运算符＞＜表达式＞  ＜循环语句＞   ::=  while '('＜条件＞')'＜语句＞| for'('＜标识符＞＝＜表达式＞;＜条件＞;＜标识符＞＝＜标识符＞(+|-)＜步长＞')'＜语句＞  ＜步长＞::= ＜无符号整数＞  ＜情况语句＞  ::=  switch ‘(’＜表达式＞‘)’ ‘{’＜情况表＞＜缺省＞‘}’  ＜情况表＞   ::=  ＜情况子语句＞{＜情况子语句＞}  ＜情况子语句＞  ::=  case＜常量＞：＜语句＞  ＜缺省＞   ::=  default :＜语句＞  ＜有返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'  ＜无返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'  ＜值参数表＞   ::= ＜表达式＞{,＜表达式＞}｜＜空＞  ＜语句列＞   ::= ｛＜语句＞｝  ＜读语句＞    ::=  scanf '('＜标识符＞')'  ＜写语句＞    ::= printf '(' ＜字符串＞,＜表达式＞ ')'| printf '('＜字符串＞ ')'| printf '('＜表达式＞')'  ＜返回语句＞   ::=  return['('＜表达式＞')']   1. 语法分析方法 2. **选用方法**   本次实验实现语法分析的思想主要是采用自顶向下的递归调用的方式去分析代码，并产生语法分析的结果。  文法每一个非终结符号对应一个递归过程，对于每个递归过程，一旦发现它的某个产生式与输入匹配，则用该产生式展开分析树。  采用自顶向下的递归调用的语法分析的第一步是改写文法，也就是保证文法是没有左递归的，然后提取左公因子，整个文法就会满足我们递归调用分析的要求。   1. **求文法的first集合**  * FIRST集合构造基于课本算法，连续使用下面的规则，直到每个集合不再增大为止： * 若X∈VT，则FIRST(X)={X}； * 若X∈VN，且有产生式X→a…，其中 a∈VT，则把a加入到FIRST(X)中； * 若X→ε也是产生式，则 ε 也加入到FIRST(X)中； * 若X→Y…是产生式，且Y∈VN，则把FIRST(Y)中的所有非ε元素加入到 FIRST(X)中； * 若X→Y1Y2…Yk是产生式，如果对某个i， FIRST(Y1)、FIRST(Y2)、…、FIRST(Yi-1)都含有ε， 即Y1Y2…Yi-1 \*⇑ ε， 则把FIRST(Yi)中的所有非ε元素加入到FIRST(X)中； * 若所有FIRST(Yi)均含有ε，其中i=1、2、…、k，则把 ε 加入到 FIRST(X)中  1. **求文法的follow集合**  * FOLLOW集合构造基于课本算法，连续使用下面的规则，直到每个集合不再增大为止： * 对文法开始符号S，置$于FOLLOW(S)中，$为输入符号串的右尾标志。 * 若A→αBβ是产生式，则把FIRST(β)中的所有非ε元素加入到 FOLLOW(B)中。 * 若A→αB是产生式，或A→αBβ是产生式，但是ε∈FIRST（β），则把FOLLOW(A) 中的所有元素加入到FOLLOW(B)中。  1. **根据first集合和follow集合编写不同的分析函数**  * 对于文法中的每个非终结符A及其任意候选式α，如有产生式A→α： * 如果当前输入符号a∈FIRST（α）时，α应被选做A的唯一合法代表去执行分析任务，即用α展开A * 如果ε∈FIRST（α），并且当前输入符号b∈FOLLOW（A）,这里b∈VT∪{$}，则A→α就认为A自动得到匹配  1. **对输入语句进行预测分析。递归预测分析算法如下**  * 输入：输入符号串w * 输出：若w属于L(G)，则输出w的最左推导，否则报告错误。 * 方法：首先调用起始非终结状态的函数，然后在每一个函数调用中，对于当前的输入符号串进行判断，然后递归调用不同的子语句执行即可（继续调用其他非终结符对应的函数）  1. 数据结构说明   因为采用的是递归调用的方法，所以基本上没有独特的数据结构。本次实验暂时没有涉及到符号表，但是函数模块需要记录是否有返回值，所以使用了C++自带的unordered\_map来表示函数名称和是否有返回值   |  | | --- | | unordered\_map<string, bool> has\_retval; |  1. 语法树及输出说明   本文输出包括输出词法分析token序列，以及调用每一个产生式时，在完成调用时输出左侧非终结符的名称作为语法分析树（子树）的根节点。可以把程序理解为一个后序遍历的过程。我们以最简单的示例代码段为例：   |  | | --- | | coNst int cONst1 = 001, const2 = -100;  const char const3 = '\_';  int change1;  char change3;  int gets1(int var1,int var2){      change1 = var1 + var2;      return (change1);  }  void main(){      printf("Hello World");      printf(gets1(10, 20));  } |   输入：词法分析的token序列   |  | | --- | | CONSTTK coNst  INTTK int  IDENFR cONst1  ASSIGN =  INTCON 001  COMMA ,  IDENFR const2  ASSIGN =  MINU -  INTCON 100  SEMICN ;  CONSTTK const  CHARTK char  IDENFR const3  ASSIGN =  CHARCON \_  SEMICN ; |   输出：   |  | | --- | | CONSTTK coNst  INTTK int  IDENFR cONst1  ASSIGN =  INTCON 001  <无符号整数>  <整数>  COMMA ,  IDENFR const2  ASSIGN =  MINU -  INTCON 100  <无符号整数>  <整数>  <常量定义>  SEMICN ;  CONSTTK const  CHARTK char  IDENFR const3  ASSIGN =  CHARCON \_  <常量定义>  SEMICN ;  <常量说明> |   得到的语法分析树如下：  常量说明   1. 错误处理   本次实验中，错误处理不是我们的实验内容，而是主要在第三个实验中进行错误处理，因此我们只需要在有可能出错的地方预留接口即可，以便在后续的实验中进行处理。   1. 程序说明 2. **语法分析类的定义，以及一些所需的数据结构，具体的含义见注释**  |  | | --- | | class yufafenxi{  private:      string curtok="";//现在的token      string unregonized="";//读取错误的token      int toknum=0;//token序列的长度      int curindex=0;//现在的token的下标      unordered\_map<string, bool> has\_retval;//函数是否有返回类型 |  1. **读取词法分析token序列的函数：**  |  | | --- | | void nexttoken(){      if(curindex<toknum){//如果还有token可以读取          curtok=token[curindex++];          fout<<token[curindex-1]<<" "<<val[curindex-1]<<endl;      }      else{//错误          curtok=unregonized;  } |  1. **查看输入符号的函数**   我们在函数seek中查看输入的token，其中参数step表示向前看的步数（当然，这个值也可以为负数，只是不需要为负数）   |  | | --- | | string seek(int step){      if(step+curindex>=0&&curindex+step-1<toknum){          return token[curindex+step-1];      }      else{          return unregonized;      }  } |  1. **进行递归调用预测分析**   整个语法分析从调用函数program开始，program主要分为四个部分，常量说明，变量说明，有（无）返回值函数定义，以及最后的main函数，每一个模块都可以有或者没有，因此需要根据输入来判断。   |  | | --- | | void program(){      nexttoken();      //常量说明      if(curtok=="CONSTTK"){          constant\_desc();          nexttoken();      }      //变量说明      if((curtok=="INTTK"||curtok=="CHARTK")&&(seek(2)=="SEMICN"||\      seek(2)=="COMMA"||seek(2)=="ASSIGN"||seek(2)=="LBRACK")){          variable\_desc();          nexttoken();      }      //函数定义      while(((curtok=="INTTK"||curtok=="CHARTK")&&seek(2)=="LPARENT")\      ||(curtok=="VOIDTK"&&seek(2)=="LPARENT"&&seek(1)!="MAINTK")){          if((curtok=="INTTK"||curtok=="CHARTK")&&seek(2)=="LPARENT"){              func\_def();              nexttoken();          }          else{              void\_func\_def();              nexttoken();          }      }      //主函数      mainfun();      fout<< "<程序>" << endl;  } |  1. 首先从常量说明开始调用，常量说明是常量定义的闭包，可以由0到n个常量定义语句组成  |  | | --- | | void constant\_desc(){      nexttoken();      //常量定义      constant\_def();      nexttoken();      //如果后面的输入是常量标记，那么也就是要继续调用常量定义，直到没有常量标记为止      while (seek(1)=="CONSTTK"){          nexttoken();          nexttoken();          constant\_def();          nexttoken();      }      fout<< "<常量说明>" << endl;  } |  1. 常量定义中，可以定义1到n个整数常量，或者字符常量  |  | | --- | | void constant\_def(){      //如果输入是int标记的话，那么说明定义的变量是整数      if(curtok=="INTTK"){          nexttoken();          nexttoken();          nexttoken();          integer();          //如果有逗号的话，说明之后还有同类型的常量需要定义          while (seek(1)=="COMMA"){              nexttoken();              nexttoken();              nexttoken();              nexttoken();              integer();          }      }      //如果输入是char标记的话，那么说明定义的变量是字符      if(curtok=="CHARTK"){          nexttoken();          nexttoken();          nexttoken();          CHAR();          //如果有逗号的话，说明之后还有同类型的常量需要定义          while (seek(1)=="COMMA"){              nexttoken();              nexttoken();              nexttoken();              nexttoken();              CHAR();          }      }      fout<< "<常量定义>" << endl;  } |  1. 常量说明结束后，会调用变量说明，变量说明是变量定义的闭包，包含0到n个变量定义语句  |  | | --- | | void variable\_desc(){      //调用变量定义      variable\_def();      nexttoken();      //如果后面还有变量要定义的话，继续调用变量定义      while ((seek(1)=="INTTK"||seek(1)=="CHARTK")&&(seek(3)=="SEMICN"||\      seek(3)=="COMMA"||seek(3)=="ASSIGN"||seek(3)=="LBRACK")){          nexttoken();          variable\_def();          nexttoken();      }      fout<<"<变量说明>" << endl;  } |  1. 变量定义分为初始化和无初始化，定义的变量包括int和char类型的普通变量，以及一维数组和二维数组，初始化的变量定义一次性只能定义一个变量，而无初始化的变量定义一次性可以定义很多个同类型的变量  |  | | --- | | void variable\_def(){      //有初始化的变量定义      if(seek(2)=="ASSIGN"||(seek(4)=="RBRACK"&&seek(5)=="ASSIGN")||(seek(4)=="RBRACK"&&seek(7)=="RBRACK"&&seek(8)=="ASSIGN")){          nexttoken();          //int或者char类型的变量          if(seek(1)=="ASSIGN"){              nexttoken();              nexttoken();              CONST();          }          //一维变量          else if(seek(4)=="ASSIGN"){              nexttoken();//[              nexttoken();//ui              unsigned\_integer();              nexttoken();//]              nexttoken();//=              nexttoken();//{              nexttoken();              CONST();              while (seek(1)=="COMMA")              {                  nexttoken();                  nexttoken();                  CONST();              }              nexttoken();//}          }          //二维变量          else if(seek(7)=="ASSIGN"){              nexttoken();//[              nexttoken();//ui              unsigned\_integer();              nexttoken();//]              nexttoken();//[              nexttoken();              unsigned\_integer();              nexttoken();//]              nexttoken();//=              nexttoken();//{              nexttoken();//{              nexttoken();              CONST();              while (seek(1)=="COMMA")              {                  nexttoken();                  nexttoken();                  CONST();              }              nexttoken();//}              while (seek(1)=="COMMA")              {                  nexttoken();                  nexttoken();                  nexttoken();                  CONST();                  while (seek(1)=="COMMA"){                      nexttoken();                      nexttoken();                      CONST();                  }                  nexttoken();              }              nexttoken();//}          }          fout<< "<变量定义及初始化>" << endl;      }      //无初始化的变量定义      else{          nexttoken();          //一维变量          if(seek(1)=="LBRACK"){              nexttoken();              nexttoken();              unsigned\_integer();              nexttoken();              //二维变量              if(seek(1)=="LBRACK"){                  nexttoken();                  nexttoken();                  unsigned\_integer();                  nexttoken();              }          }          //如果输入符号是逗号，说明还有无初始化的变量需要定义          while (seek(1)=="COMMA"){              nexttoken();              nexttoken();              if(seek(1)=="LBRACK"){                  nexttoken();                  nexttoken();                  unsigned\_integer();                  nexttoken();                  if(seek(1)=="LBRACK"){                      nexttoken();                      nexttoken();                      unsigned\_integer();                      nexttoken();                  }              }          }          fout<< "<变量定义无初始化>" << endl;      }      fout<< "<变量定义>" << endl;  } |  1. 之后是函数的定义，分为有返回值和无返回值两种函数的定义，两种定义都包括返回值类型和函数名称的声明，参数表和函数的语句，首先是有返回值的函数定义：  |  | | --- | | void func\_def(){      //声明头部，即返回值类型和函数名称      declaration\_header();      nexttoken();      if (seek(1)!="RPARENT"){          nexttoken();          //定义参数表          parameters\_table();          nexttoken();      }else{          fout<< "<参数表>" << endl;          nexttoken();      }      nexttoken();      nexttoken();      //函数的语句      compound\_statement();      nexttoken();      fout<< "<有返回值函数定义>" << endl;  } |  1. 其次是无返回值函数的定义：  |  | | --- | | void void\_func\_def(){      nexttoken();      //全局记录一下当前的函数，是否有返回值      has\_retval.insert(pair<string, bool>(val[curindex-1], false));      nexttoken();      //定义参数表，有可能为空      if (seek(1)!="RPARENT"){          nexttoken();          parameters\_table();          nexttoken();      }else{          fout<< "<参数表>" << endl;          nexttoken();      }      nexttoken();      nexttoken();      //函数的语句      compound\_statement();      nexttoken();      fout<< "<无返回值函数定义>" << endl;  } |  1. 两个函数的定义中都用到了复合语句，复合语句主要由常量说明，变量说明和语句列组成：  |  | | --- | | void compound\_statement(){      //常量说明      if(curtok=="CONSTTK"){          constant\_desc();          nexttoken();      }      //变量说明      if(curtok=="CHARTK"||curtok=="INTTK"){          variable\_desc();          nexttoken();      }      //语句列      statement\_column();      fout<< "<复合语句>" << endl;  } |  1. 语句列本质就是各种语句的集合  |  | | --- | | void statement\_column(){      //语句列不空      if (curtok!= "RBRACE") {          //只要有语句就一直分析下去          while (seek(1)!="RBRACE"&&curindex<toknum  ||curtok=="SEMICN") {              statement();              if (seek(1)!= "RBRACE"&&curindex<toknum) {                  nexttoken();              }              else if(seek(1)== "RBRACE"){                  break;              }          }      }      fout<< "<语句列>" << endl;  } |  1. 语句有很多种类，包括循环语句、条件语句、有返回值函数调用语句、无返回值函数调用语句、赋值语句、读语句、写语句、情况语句、返回语句，此处展示一下语句列和语句的调用函数，子语句就不过多展示了：  |  | | --- | | void statement(){      //条件语句      if(curtok=="IFTK"){          condition\_statement();      }      //循环语句      else if(curtok=="WHILETK"||curtok=="FORTK"){          loop\_statement();      }      //语句列      else if(curtok=="LBRACE"){          if(seek(1)=="RBRACE"){              fout<< "<语句列>" << endl;              nexttoken();          }          else{              nexttoken();              statement\_column();              nexttoken();          }      }      else if(curtok=="IDENFR"){          //有返回值函数调用          if(has\_retval[val[curindex-1]]){              funcall();              nexttoken();          }          //赋值语句          else if(seek(1)=="ASSIGN"||seek(1)=="LBRACK"){              assign\_statement();              nexttoken();          }          //无返回值函数调用语句          else if(!has\_retval[val[curindex-1]]){              voidfuncall();              nexttoken();          }        }      //读语句      else if(curtok=="SCANFTK"){          read\_statement();          nexttoken();      }      //写语句      else if(curtok=="PRINTFTK"){          write\_statement();          nexttoken();      }      //返回语句      else if(curtok=="RETURNTK"){          return\_statement();          nexttoken();      }      //情况语句      else if(curtok=="SWITCHTK"){          case\_statement();      }      //空      else if(curtok=="SEMICN"){          //nothing      }      fout<< "<语句>" << endl;  } |  1. 程序的最后一个板块是主函数，主函数的内容就是复合语句：  |  | | --- | | void mainfun(){      nexttoken();      nexttoken();      nexttoken();      nexttoken();      nexttoken();      //直接是复合语句      compound\_statement();      nexttoken();      fout<< "<主函数>" << endl;  } |  1. 错误处理：同前文描述，错误主要是预留接口，随机找一处错误处理接口展示：  |  | | --- | | //空  else if(curtok=="SEMICN"){      //nothing  }  else{      //错误处理接口  }  fout<< "<语句>" << endl; | | | | | | |
| 1. 实验测试结果 2. 本地开发时的测试用例 3. **代码**  |  | | --- | | coNst int cONst1 = 001, const2 = -100;  const char const3 = '\_';  int change1;  char change3;  int gets1(int var1,int var2){      change1 = var1 + var2;      return (change1);  }  void main(){      printf("Hello World");      printf(gets1(10, 20));  } |  1. **结果**  |  | | --- | | CONSTTK coNst  INTTK int  IDENFR cONst1  ASSIGN =  INTCON 001  <无符号整数>  <整数>  COMMA ,  IDENFR const2  ASSIGN =  MINU -  INTCON 100  <无符号整数>  <整数>  <常量定义>  SEMICN ;  CONSTTK const  CHARTK char  IDENFR const3  ASSIGN =  CHARCON \_  <常量定义>  SEMICN ;  <常量说明>  INTTK int  IDENFR change1  <变量定义无初始化>  <变量定义>  SEMICN ;  CHARTK char  IDENFR change3  <变量定义无初始化>  <变量定义>  SEMICN ;  <变量说明>  INTTK int  IDENFR gets1  <声明头部>  LPARENT (  INTTK int  IDENFR var1  COMMA ,  INTTK int  IDENFR var2  <参数表>  RPARENT )  LBRACE {  IDENFR change1  ASSIGN =  IDENFR var1  <因子>  <项>  PLUS +  IDENFR var2  <因子>  <项>  <表达式>  <赋值语句>  SEMICN ;  <语句>  RETURNTK return  LPARENT (  IDENFR change1  <因子>  <项>  <表达式>  RPARENT )  <返回语句>  SEMICN ;  <语句>  <语句列>  <复合语句>  RBRACE }  <有返回值函数定义>  VOIDTK void  MAINTK main  LPARENT (  RPARENT )  LBRACE {  PRINTFTK printf  LPARENT (  STRCON Hello World  <字符串>  RPARENT )  <写语句>  SEMICN ;  <语句>  PRINTFTK printf  LPARENT (  IDENFR gets1  LPARENT (  INTCON 10  <无符号整数>  <整数>  <因子>  <项>  <表达式>  COMMA ,  INTCON 20  <无符号整数>  <整数>  <因子>  <项>  <表达式>  <值参数表>  RPARENT )  <有返回值函数调用语句>  <因子>  <项>  <表达式>  RPARENT )  <写语句>  SEMICN ;  <语句>  <语句列>  <复合语句>  RBRACE }  <主函数>  <程序> |  1. 平台测试 2. **结果**   **QQ截图20220423194702** | | | | | |
| 1. 实验总结 2. BUG记录 3. **BUG1** 4. 错误原因   在语句推导的时候，在测试平台上语句列有可能是全空，也有可能是有内容（文法定义不是全空，至少有一个分号）在代码中需要分情况讨论   1. 解决方法：只需要在代码中分类即可  |  | | --- | | else if(curtok=="LBRACE"){      if(seek(1)=="RBRACE"){          fout << "<语句列>" << endl;          nexttoken();      }      else{          nexttoken();          statement\_column();          nexttoken();      }  } |  1. **BUG2** 2. 错误原因   在函数声明头部的时候，错把token类别码记录了，没有记录单词值，因此在有函数调用语句的时候就会出错   1. 解决方法：把值换成val[curindex-1]  |  | | --- | | has\_retval.insert(pair<string,bool>(val[curindex-1],true)); |  1. **BUG3** 2. 错误原因   有一些函数调用时忘记添加对应的语法分析输出，导致平台无法通过   1. 解决方法   添加对应的语法分析输出语句即可，例如   |  | | --- | | fout << "<声明头部>" << endl; |  1. **BUG4** 2. 错误原因   最开始在变量定义的时候未考虑二维数组，测试平台上无法识别   1. 解决方法   在变量定义函数中增加对应的识别逻辑，如果是二维数组的话，后面的第七个token一定是]，第八个是=   |  | | --- | | if(seek(2)=="ASSIGN"||(seek(4)=="RBRACK"&&seek(5)=="ASSIGN")||  (seek(4)=="RBRACK"&&seek(7)=="RBRACK"&&seek(8)=="ASSIGN")){ |  1. **BUG5** 2. 错误原因   最开始的时候直接定义各个函数，因为存在互相调用的关系，编译器会报错，没有声明函数   1. 解决方法   后面将语法分析整个定义为类，然后把递归调用的函数都写为类内函数，如下   |  | | --- | | class yufafenxi{  private:      string curtok="";      string unregonized="";      int toknum=0;      int curindex=0;      unordered\_map<string, bool> has\_retval;      ofstream fout;  public:      void nexttoken(){}      void unsigned\_integer(){} |  1. 心得体会   本次实验难度尚可，相较于实验一工程量要大了很多。但整体上可以参考书上的算法和伪代码，只需要把相应算法实现即可。不过在完成代码的过程中遇到的各种bug也还是十分令人头疼，所幸有测试文件供我们下载下来debug。 | | | | | |