**《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 | 错误处理作业 | | | | |
| 实验时间 | 2022.4.23 | | 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： | | | | | |
| 1. 实验目的 2. 掌握语法制导定义和语法翻译方案 3. 编写语义分析程序，根据给定的文法设计并实现错误处理程序，能诊察出常见的语法和语义错误，进行错误局部化处理，并输出错误信息。 | | | | | |
| 1. 实验内容   请根据给定的文法设计并实现错误处理程序，能诊察出常见的语法和语义错误，进行错误局部化处理，并输出错误信息。为方便自动评测，输入输出及处理要求如下：   1. 输入的被编译源文件统一命名为testfile.txt；错误信息输出到命名为error.txt的结果文件中； 2. 结果文件中包含如下两种信息：错误所在的行号 错误的类别码 （行号与类别码之间只有一个空格，类别码严格按照表格中的小写英文字母）其中错误类别码按下表中的定义输出，行号从1开始计数：  |  |  |  | | --- | --- | --- | | **错误类型** | **错误类别码** | **解释及举例** | | 非法符号或不符合词法 | a | 例如字符与字符串中出现非法的符号，符号串中无任何符号 | | 名字重定义 | b | 同一个作用域内出现相同的名字（不区分大小写） | | 未定义的名字 | c | 引用未定义的名字 | | 函数参数个数不匹配 | d | 函数调用时实参个数大于或小于形参个数 | | 函数参数类型不匹配 | e | 函数调用时形参为整型，实参为字符型；或形参为字符型，实参为整型 | | 条件判断中出现不合法的类型 | f | 条件判断的左右表达式只能为整型，其中任一表达式为字符型即报错，例如’a’==1 | | 无返回值的函数存在不匹配的return语句 | g | 无返回值的函数中可以没有return语句，也可以有形如return;的语句，若出现了形如return(表达式);或return();的语句均报此错误 | | 有返回值的函数缺少return语句或存在不匹配的return语句 | h | 例如有返回值的函数无任何返回语句；或有形如return;的语句；或有形如return();的语句；或return语句中表达式类型与返回值类型不一致 | | 数组元素的下标只能是整型表达式 | i | 数组元素的下标不能是字符型 | | 不能改变常量的值 | j | 这里的常量指的是声明为const的标识符。例如 const int a=1;在后续代码中如果出现了修改a值的代码，如给a赋值或用scanf获取a的值，则报错。 | | 应为分号 | k | 应该出现分号的地方没有分号，例如int x=1缺少分号 （7种语句末尾，for语句中，常量定义末尾，变量定义末尾） | | 应为右小括号’)’ | l | 应该出现右小括号的地方没有右小括号，例如fun(a,b;，缺少右小括号（有/无参数函数定义，主函数，带括号的表达式，if，while，for，switch，有/无参数函数调用，读、写、return） | | 应为右中括号’]’ | m | 应该出现右中括号的地方没有右中括号，例如int arr[2;缺少右中括号（一维/二维数组变量定义有/无初始化，因子中的一维/二维数组元素，赋值语句中的数组元素） | | 数组初始化个数不匹配 | n | 任一维度的元素个数不匹配，或缺少某一维的元素即报错。  例如int a[2][2]={{1,2,3},{1,2}} | | <常量>类型不一致 | o | 变量定义及初始化和switch语句中的<常量>必须与声明的类型一致。int x=’c’;  int y;  switch(y){  case(‘1’) ….  } | | 缺少缺省语句 | p | switch语句中，缺少<缺省>语句。 |  1. 所有错误都不会出现恶意换行的情况，包括字符、字符串中的换行符、函数调用等等。 2. 其他类型的错误，错误的行号以能够断定发现出错的一个符号的行号为准。例如有返回值的函数缺少返回语句的错误，只有当识别到函数末尾的}时仍未出现返回语句，才可以断定出错，报错行号即为}的行号。   【输入形式】testfile.txt中的存在语法或语义错误的测试程序。  【输出形式】按如上要求将错误处理结果输出至error.txt中。  【评分标准】  本次作业的每个测试程序各包含1-3个错误，均来自上表；若一个测试程序中包含多个错误，准确报出第一个错误能得到60%的分数，其余错误则按实报错误占应报错误的比例得分  【特别提醒】   1. 上表中只列举了部分错误类型和报告该错误类型的情况，未包含的错误类型或错误情况，需要自行设计，本次作业考核不涉及； 2. 完成本次作业时，请勿输出词法分析和语法分析作业要求输出的内容； 3. 本次考核之外，发现错误时最好直接输出描述信息，而不是仅给出错误类别码。 4. **每一行中最多只有一个错误。**   【参考资料】根据PASCAL-S文法的定义，阅读编译器源代码，了解符号表的设计实现方案和错误处理实现方案；在此基础上，为自己的编译器添加符号表管理、错误处理功能（包括语法错误和语义错误），编译器源代码见[pascals-compiler.docx](https://course.educg.net/userfiles/file/2021/16157911116670131914646.docx" \o "pascals-compiler.docx)。  【开发语言及环境】用C/C++实现，平台支持C++11标准，源代码文件必须使用UTF-8编码，才能够输出评测平台能够识别的中文(如果不确定源文件的编码，推荐使用notepad++查看)  【辅助工具】[错误处理测试程序库.zip](https://course.educg.net/userfiles/file/2021/16157911588540220464646.zip" \o "错误处理测试程序库.zip)  【提交形式】将所开发的语法分析程序的源文件（.cpp/.c/.h，不含工程文件）打包为zip或rar后提交（注意在MAC下压缩会产生额外的文件到压缩包中，需删掉额外文件后提交），实验报告一起打包提交。 | | | | | |
| 1. 实验过程或算法（源程序） 2. 实验概述   在本次实验中，我完成了一个语义分析和错误处理程序，可以对每个产生式做常见的错误检查与处理。错误检查主要包括一些终结符的缺失如( , { , ; 等，类型的检查，包括名字重定义，未定义的名字，不能改变常量的值等等。  本次实验主要基于语法分析器的代码之上，使用递归调用的方法来对语句做语法分析，同时执行相应的错误检查和恢复。  为了便于实现，我们定义了数组line用来标识当前token的行数，便于定位源代码的错误位置。  同样，我还定义了符号表类，存放符号表的栈，以及一个指向全局符号表的指针，用来辅助进行语义分析。   1. 数据结构说明 2. 符号表  |  | | --- | | class attributes{  public:  *//类型*      string type;  *//引用行*      vector<int> quote\_lines;  *//声明行*      int declaim\_line;  *//函数参数类型表*      vector<string> parameter\_types;  *//是否有返回值*      bool has\_retval;  *//是否是常量*      bool is\_const=false;  }; |  1. 存放符号表的栈  |  | | --- | | stack<unordered\_map<string,attributes>\*>symbol\_table; |  1. 全局符号表指针  |  | | --- | | unordered\_map<string,attributes>\*global\_symbol\_table; |  1. 其他语义分析全局变量  |  | | --- | | int is\_def\_func =0;  string def\_func\_type=""; |  1. 程序说明 2. 非法符号或不符合词法  |  | | --- | | void CHAR(){  *//符合要求的字符*      if(val[curindex-1]=="+"||val[curindex-1]=="-"||val[curindex-1]=="\*"||val[curindex-1]=="/"\      ||val[curindex-1]=="\_"||isalpha(int(val[curindex-1][0]))||isdigit(int(val[curindex-1][0]))){        }  *//不符合要求的字符*      else{          fout<<row[curindex-1]<<" "<<"a"<<endl;      }  }  void str(){      if(curtok=="STRCON"){  *//为空*          if(val[curindex-1]==""){              fout<<"a"<<" "<<row[curindex-1]<<endl;          }          else{              for(auto &c:val[curindex-1]){                  if(int(c)!=34&&int(c)<=126&&int(c)>=32){                  }  *//出现不符合要求的字符*                  else{                      fout<<row[curindex-1]<<" "<<"a"<<endl;                      break;                  }              }          }      }  } |  1. 名字重定义   在常量定义、变量定义、声明头部、参数表定义、无返回值函数定义中有可能会发生名字重定义，基本思路就是输出错误，不把变量加入符号表，然后继续分析，如下展示了常量定义情况下的部分处理代码。   |  | | --- | | 常量定义  *//常量定义*  void constant\_def(){      if(curtok=="INTTK"){          nexttoken();  *//如果符号表里有当前符号，那么就是重定义*          if(symbol\_table.top()->count(val[curindex-1])){              b\_error();          } |  1. 未定义的名字   在因子，赋值语句，for循环，读语句，语句文法中都有可能出现引用未定义的符号这种情况，基本处理思路就是识别错误后，当成一个普通的变量继续执行语法分析，如下展示了因子文法中的部分处理代码。   |  | | --- | | 因子文法中的变量有可能未定义  *//因子*  void factor(){  if(curtok=="IDENFR"){  *//如果在全局符号表和当前符号表中都没有找到符号，那么就是引用未定义的名字*          if(!symbol\_table.top()->count(val[curindex-1])&&\          !has\_retval.count(val[curindex-1])&&\          !global\_symbol\_table->count(val[curindex-1])){              c\_error();          } |  1. 函数参数个数不匹配   即函数调用时实参个数大于或小于形参个数，只会在函数调用的时候发生，有无返回值函数调用处理方法相同，都是用记录的参数个数与实际的参数个数比较，不匹配则报错，如下展示了每一种情况下的部分处理代码。   |  | | --- | | 参数表中处理  *//参数表*  void value\_par\_table(){      unordered\_map<string,attributes> gst=\*global\_symbol\_table;      int count1=parameters[val[curindex-3]].size();      string namet=val[curindex-3];      int count2=0;      vector<string>p1;      vector<string>p2;  *//p1=gst[val[curindex-3]].parameter\_types;*      p1=(\*global\_symbol\_table)[val[curindex-3]].parameter\_types;      p2.push\_back(express());      count2++;      while (seek(1)=="COMMA")      {          nexttoken();          nexttoken();          p2.push\_back(express());          count2++;      }  *//count1是当前函数的参数个数，count2是实际调用时的参数个数*  *//如果不相同的话，那么就是参数个数不匹配错误*      if(count1!=count2){          d\_error();      } | | 调用时没有参数，直接在函数调用分析时处理  *//函数调用*  void funcall(){      nexttoken();      if(seek(1)!="RPARENT"){          nexttoken();          value\_par\_table();          nexttoken();      }  *//没有参数*      else{  *//看符号表中记录的是不是也没有参数*  *//实际没有，记录的有，就报错*          if(parameters[val[curindex-2]].size()!=0){              d\_error();          }          nexttoken();      }  } |  1. 函数参数类型不匹配   即函数调用时形参为整型，实参为字符型；或形参为字符型，实参为整型，只会在函数调用的时候发生，基本处理思路为比较每一个参数的类型是否一致，如果不一致，报错，如下展示了部分处理代码。   |  | | --- | | *//值参数表*  void value\_par\_table(){      unordered\_map<string,attributes> gst=\*global\_symbol\_table;      int count1=parameters[val[curindex-3]].size();      string namet=val[curindex-3];      int count2=0;      vector<string>p1;      vector<string>p2;      p1=(\*global\_symbol\_table)[val[curindex-3]].parameter\_types;      p2.push\_back(express());      count2++;      while (seek(1)=="COMMA")      {          nexttoken();          nexttoken();          p2.push\_back(express());          count2++;      }      if(count1!=count2){          d\_error();      }  *//比较每一个参数的类型是否一致，如果不一致，报错*      else{          for(int i=0;i<p2.size();i++){              if(p1[i]!=p2[i]){                  e\_error();                  break;              }          }      }  } |  1. 条件判断中出现不合法的类型   即条件判断的左右表达式只能为整型，其中任一表达式为字符型即报错，例如’a’==1，只会条件调用的时候发生，基本处理思路为比较表达式两边的类型是否一致，如果不一致，报错，如下展示了部分处理代码。   |  | | --- | | *//条件*  void conditon(){      string type1,type2;      type1=express();        if(seek(1)=="LSS"||seek(1)=="LEQ"||seek(1)=="GRE"||seek(1)=="GEQ"||seek(1)=="EQL"||seek(1)=="NEQ"){          nexttoken();          nexttoken();          type2=express();      }  *//type1，2分别是表达式左右两段的类型*  *//如果不一致的话，报错*      if(type1!="INTTK"||type2!="INTTK"){          f\_error();      }  } |  1. 无返回值的函数存在不匹配的return语句   即无返回值的函数中可以没有return语句，也可以有形如return;的语句，若出现了形如return(表达式);或return();的语句则报错，基本思路就是在定义的时候记录一下状态（有无返回值函数定义），如果出现了有值的返回语句，而此时在定义无返回值的函数，那么就报错。如下展示了部分处理代码。   |  | | --- | | *//＜返回语句＞ ::= return['('＜表达式＞')']*  void return\_statement(){  *//如果有左括号，表示有返回内容，不是无返回值的函数*      if(seek(1)=="LPARENT"){  *//is\_def\_func==2表示是无返回值的函数*  *//无返回值函数有返回内容，错误*          if(is\_def\_func==2){              g\_error();          } |  1. 有返回值的函数缺少return语句或存在不匹配的return语句   如有返回值的函数无任何返回语句；或有形如return;的语句；或有形如return();的语句；或return语句中表达式类型与返回值类型不一致，基本思路就是在定义的时候记录一下状态（有无返回值函数定义），如果出现了诸如上述的返回语句，而此时在定义有返回值的函数，那么就报错。如下展示了部分处理代码。   |  | | --- | | 返回类型不一致和无返回值的情况  *//＜返回语句＞ ::= return['('＜表达式＞')']*  void return\_statement(){  *//如果有左括号，表示有返回内容，不是无返回值的函数*      if(seek(1)=="LPARENT"){  *//is\_def\_func==2表示是无返回值的函数*  *//无返回值函数有返回内容，错误*          if(is\_def\_func==2){              g\_error();          }          nexttoken();          nexttoken();          string type1=express();  *//return语句中表达式类型与返回值类型不一致*          if(type1!=def\_func\_type){              h\_error();          }          nexttoken();      }  *//如果没有括号，表示无返回内容，是无返回值的函数*      else{  *//此时如果在定义有返回值的函数，就报错*          if(is\_def\_func==1){              h\_error();          }      }  *//重置定义标记*      is\_def\_func=0;  } | | 没有返回语句的情况  *//有返回值函数定义*  void func\_def(){      is\_def\_func=1;      declaration\_header();      vector<string> tempvec;      parameters.emplace(val[curindex-1],tempvec);      nexttoken();      symbol\_table.push(new unordered\_map<string,attributes>());      if (seek(1)!="RPARENT"){          nexttoken();          parameters\_table();          nexttoken();      }else{          nexttoken();      }      nexttoken();      nexttoken();      compound\_statement();      symbol\_table.pop();      nexttoken();  *//没有返回语句*      if(is\_def\_func==1){          h\_error();      }      is\_def\_func=0;      def\_func\_type="";  } |  1. 数组元素的下标只能是整型表达式   数组元素的下标不能是字符型，在赋值语句和因子中会出现数组变量，有可能会出现字符下标，基本处理思路为检查每一个表达式类别，如果不为整数型的话，报错。如下展示了部分处理代码。   |  | | --- | | 赋值语句  void assign\_statement(){  *//数组变量*      else{          nexttoken();          string type1=express();  *//一维下标是否是无符号整数*          if(type1=="CHARTK"){              i\_error();          }          m\_error();          if(seek(1)=="ASSIGN"){              nexttoken();              nexttoken();              express();          }          else{*//seek(1)="["*              nexttoken();              nexttoken();              string type2=express();  *//二维下标是否是无符号整数*              if(type2=="CHARTK"){                  i\_error();              }              m\_error();              nexttoken();              nexttoken();              express();          }      }  } | | 因子  *//因子*  void factor(){      if(curtok=="IDENFR"){          if(!symbol\_table.top()->count(val[curindex-1])&&\          !has\_retval.count(val[curindex-1])&&\          !global\_symbol\_table->count(val[curindex-1])){              c\_error();          }          if(has\_retval[val[curindex-1]]){              funcall();          }  *//数组变量*          else if(seek(1)=="LBRACK"){              nexttoken();              nexttoken();              string type=express();  *//一维下标是否是无符号整数*              if(type=="CHARTK"){                  i\_error();              }              m\_error();              if(seek(1)=="LBRACK"){                  nexttoken();                  nexttoken();                  string type2=express();  *//二维下标是否是无符号整数*                  if(type2=="CHARTK"){                      i\_error();                  }                  m\_error();              }          }      } |  1. 不能改变常量的值   这里的常量指的是声明为const的标识符。例如 const int a=1;在后续代码中如果出现了修改a值的代码，如给a赋值，在for语句中使用常量来递增循环，或用scanf获取a的值，则报错。基本处理思路为每次在赋值的时候，判断当前变量是否是常量，如果是则报错，如下展示了部分处理代码。   |  | | --- | | 赋值语句  *//赋值语句*  void assign\_statement(){      if(!symbol\_table.top()->count(val[curindex-1])&&!global\_symbol\_table->count(val[curindex-1])){          c\_error();      }      bool is\_const=false;      if(symbol\_table.top()->count(val[curindex-1])){          is\_const=(\*symbol\_table.top())[val[curindex-1]].is\_const;      }      else if(global\_symbol\_table->count(val[curindex-1])){          is\_const=(\*global\_symbol\_table)[val[curindex-1]].is\_const;      }  *//如果变量是常量直接报错*      if(is\_const==true){          j\_error();      } | | For语句  *//＜循环语句＞*  void loop\_statement(){      else if(curtok=="FORTK"){          nexttoken();          nexttoken();          if(!symbol\_table.top()->count(val[curindex-1])&&!global\_symbol\_table->count(val[curindex-1])){              c\_error();          }  *//是不是常量*          bool is\_const=false;          if(symbol\_table.top()->count(val[curindex-1])){              is\_const=(\*symbol\_table.top())[val[curindex-1]].is\_const;          }          else if(global\_symbol\_table->count(val[curindex-1])){              is\_const=(\*global\_symbol\_table)[val[curindex-1]].is\_const;          }  *//如果是常量赋值直接报错*          if(is\_const==true){              j\_error();          } | | 读语句  *//read statement*  void read\_statement(){      nexttoken();      nexttoken();      if(!symbol\_table.top()->count(val[curindex-1])&&!global\_symbol\_table->count(val[curindex-1])){          c\_error();      }      bool is\_const=false;      if(symbol\_table.top()->count(val[curindex-1])){          is\_const=(\*symbol\_table.top())[val[curindex-1]].is\_const;      }      else if(global\_symbol\_table->count(val[curindex-1])){          is\_const=(\*global\_symbol\_table)[val[curindex-1]].is\_const;      }  *//如果是常量赋值直接报错*      if(is\_const==true){          j\_error();      }      nexttoken();  } |  1. 应为分号   应该出现分号的地方没有分号，例如int x=1缺少分号 （在7种语句末尾，for语句，常量定义末尾，变量定义末尾都有可能出现）。基本处理思路为读到该出现分号的时候，判断当前token是否是分号，如果不是则报错，然后不跳过当前token，用它继续进行分析，如下展示了部分处理代码。   |  | | --- | | 常量定义完成时，没有分号就报错，不跳过继续分析；有则跳过他继续分析  *//常量说明*  void constant\_desc(){      nexttoken();      constant\_def();      if(seek(1)=="SEMICN"){          nexttoken();      }  *//没有分号报错*      else{          k\_error();      } |  1. 应为右小括号’)’   应该出现右小括号的地方没有右小括号，例如fun(a,b;，缺少右小括号（在有/无参数函数定义，主函数，带括号的表达式，if，while，for，switch，有/无参数函数调用，读、写、return中都有可能出现），基本处理思路和缺少分号相同，可参见第11条，如下展示了部分处理代码。   |  | | --- | | 无返回值函数调用中的值参数表的右括号缺失判断  *//无返回值函数调用*  void voidfuncall(){      nexttoken();      if(seek(1)=="PLUS"||seek(1)=="MINU"||seek(1)=="IDENFR"\      ||seek(1)=="LPARENT"||seek(1)=="INTCON"||seek(1)=="CHARCON"){          nexttoken();          value\_par\_table();          nexttoken();      }      else{          if(parameters[val[curindex-2]].size()!=0){              d\_error();          }  *//缺少右括号*          if(seek(1)!="RPARENT"){              l\_error();          }          else{              nexttoken();          }      }  } |  1. 应为右中括号’]’   应该出现右中括号的地方没有右中括号，例如int arr[2;缺少右中括号（在一维/二维数组变量定义有/无初始化，因子中的一维/二维数组元素，赋值语句中的数组元素中都可能出现），基本处理思路和缺少分号相同，可参见第11条，如下展示了部分处理代码。   |  | | --- | | void m\_error(){  *//如果有右中括号*      if(seek(1)=="RBRACK"){          nexttoken();      }  *//没有，报错*      else{          fout<<row[curindex-1]<<" "<<"m"<<endl;      }  } |  1. 数组初始化个数不匹配   任一维度的元素个数不匹配，或缺少某一维的元素即报错。例如int a[2][2]={{1,2,3},{1,2}}，基本处理思路为，在变量定义时就记录下维度的长度，然后在后续初始化的时候，记录每一个维度初始化的长度，最后和记录的比较，不同则报错，如下展示了部分处理代码。   |  | | --- | | *// 变量定义*  void variable\_def(){      else if(seek(4)=="ASSIGN"){          nexttoken();*//[*          nexttoken();*//ui*          unsigned\_integer();  *//定义的维度长度*          int def\_size=stoi(val[curindex-1]);  *//实际的维度长度*          int act\_size=0;          m\_error();          nexttoken();*//=*          nexttoken();*//{*          nexttoken();          ttype=CONST();          if(ttype!=type){              is\_const\_same=false;          }          ttype="";          act\_size++;          while (seek(1)=="COMMA")          {              nexttoken();              nexttoken();              ttype=CONST();              if(ttype!=type){                  is\_const\_same=false;              }              ttype="";  *//实际的维度长度++*              act\_size++;          }          nexttoken();*//}*  *//如果记录的维度长度和初始化的维度长度不一样，则报错*          if(def\_size!=act\_size){              n\_error();          }      } |  1. 常量类型不一致   变量定义及初始化和switch语句中的<常量>必须与声明的类型一致。错误例如：   |  | | --- | | int x=’c’;  int y;  switch(y){  case(‘1’) ….  } |   基本处理思路为，在变量定义时就记录下类别，然后在后续初始化的时候，把每一个常量的类型和标识符的比较，不同则报错，如下展示了部分处理代码。   |  | | --- | | *//情况语句*  void case\_statement(){      nexttoken();      nexttoken();  *//case的变量的类别*      string type1=express();      nexttoken();      nexttoken();      nexttoken();  *//在情况表中判断不同情况的常量是不是和变量的类别一致*      bool is\_const\_same=fact\_sheet(type1);      nexttoken();      if(curtok=="DEFAULTTK"){          deft();      }      else{          p\_error();      }      nexttoken();  *//如果不是的话，直接报错*      if(!is\_const\_same){          o\_error();      }  } |  1. 缺少缺省语句   switch语句中，缺少<缺省>语句。基本处理思路为，在情况语句期望出现缺省语句的时候，判断当前token是不是default，不是则报错，如下展示了部分处理代码。   |  | | --- | | *//情况语句*      void case\_statement(){  *//期望出现缺省语句*          if(curtok=="DEFAULTTK"){              deft();          }  *//没有出现缺省语句，报错*          else{              p\_error();          }      } | | | | | | |
| 1. 实验结果及分析和（或）源程序调试过程 2. 本地测试 3. 测试程序  |  | | --- | | const int const1 = 1, const2 = -100;  const char const3 = '?';  int change1;  char change3;  int gets1(int var1,int var2){  change1 = var1 + var2 return (change1);  }  void main(){  change1 = 10;  printf("Hello World");  printf(gets1(10, 20));  } |  1. 结果  |  | | --- | | 2 a  6 k |  1. 平台测试 2. 结果   平台 | | | | | |
| 1. 实验总结 2. BUG记录 3. BUG1 4. 问题   在使用符号表的时候，因为提前做实验的缘故，完全没有考虑到块结构的存在，导致在引用全局变量的时候会出现未定义的名字这样的错误。   1. 解决方法   用栈结构来保存不同的符号表，即可完美的解决块结构的问题   |  | | --- | | stack<unordered\_map<string,attributes>\*>symbol\_table; |  1. BUG2 2. 问题   在处理有返回值的return语句时，只考虑return语句不符合要求的情况，根本没考虑缺少return语句的情况。   1. 解决方法   添加处理无return语句的逻辑，通过一个全局变量标记是不是在定义有返回值的函数，然后记录在定义时是否出现return语句，如果没有return语句，报错。   1. BUG3 2. 问题   在常量值不可修改错误中，忽视了scanf语句也可以给常量赋值的作用   1. 解决方法   在scanf语句加入错误处理逻辑，如果赋值的变量是一个常量，直接报错   |  | | --- | | *//read statement*      void read\_statement(){          nexttoken();          nexttoken();          bool is\_const=false;          if(symbol\_table.top()->count(val[curindex-1])){              is\_const=(\*symbol\_table.top())[val[curindex-1]].is\_const;          }          else if(global\_symbol\_table->count(val[curindex-1])){              is\_const=(\*global\_symbol\_table)[val[curindex-1]].is\_const;          }  *//如果是常量赋值直接报错*          if(is\_const==true){              j\_error();          }          nexttoken();      } |  1. BUG4 2. 问题   在处理数组初始化个数不匹配的错误时，面对二维数组，我只是通过括号的个数来处理了维度是否有问题，而没有处理每一个一维数组个数是不是匹配   1. 解决方法   用一个数组vector<int>来记录每一个一维初始化的长度，然后通过调用size函数，先检查二维的长度是否正确，然后检查每一个一维的长度是否正确。   1. 心得   本次实验难度也不太大，主要是需要理解语义分析的实现。我们只需要定义符号表这个数据结构，配上一些报告之前提到的思路即可完成大多数实验要求，然后用栈来存放符号表就能解决局部变量和全局变量的问题。书上对于自顶向下的递归调用方法介绍的并不多，只能自己一步一步摸索，而且一个错误可能出现在很多分散的地方，需要考虑得非常周全才可以通过测试用例，因此本次实验耗时还是比较长，完成的时候也非常痛苦。 | | | | | |