【问题描述】

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。为了方便自动评测，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 | 对应文法及出错符号(…省略该条规则后续部分) |
| --- | --- | --- | --- |
| 非法符号 | a | 格式字符串中出现非法字符报错行号为**<FormatString>**所在行数。 | <FormatString> → ‘“‘{<Char>}’”’ |
| 名字重定义 | b | 函数名或者变量名在**当前作用域**下重复定义。注意，变量一定是同一级作用域下才会判定出错，不同级作用域下，内层会覆盖外层定义。报错行号为**<Ident>**所在行数。 | <ConstDef>→<Ident> …  <VarDef>→<Ident> … |<Ident> …  <FuncDef>→<FuncType><Ident> …  <FuncFParam> → <BType> <Ident> ... |
| 未定义的名字 | c | 使用了未定义的标识符报错行号为**<Ident>**所在行数。 | <LVal>→<Ident> …  <UnaryExp>→<Ident> … |
| 函数参数个数不匹配 | d | 函数调用语句中，参数个数与函数定义中的参数个数不匹配。报错行号为函数调用语句的**函数名**所在行数。 | <UnaryExp>→<Ident>‘(’[FuncRParams ]‘)’ |
| 函数参数类型不匹配 | e | 函数调用语句中，参数类型与函数定义中对应位置的参数类型不匹配。报错行号为函数调用语句的**函数名**所在行数。 | <UnaryExp>→<Ident>‘(’[FuncRParams ]‘)’ |
| 无返回值的函数存在不匹配的return语句 | f | 报错行号为**‘return’**所在行号。 | <Stmt>→‘return’ {‘[’Exp’]’}‘;’ |
| 有返回值的函数缺少return语句 | g | 只需要考虑函数末尾是否存在return语句，**无需考虑数据流**。报错行号为函数**结尾的’}’**所在行号。 | FuncDef → FuncType Ident ‘(’ [FuncFParams] ‘)’ Block  MainFuncDef → 'int' 'main' '(' ')' Block |
| 不能改变常量的值 | h | <LVal>为常量时，不能对其修改。报错行号为**<LVal>**所在行号。 | <Stmt>→<LVal>‘=’ <Exp>‘;’|<LVal>‘=’ ‘getint’ ‘(’ ‘)’ ‘;’ |
| 缺少分号 | i | 报错行号为分号**前一个非终结符**所在行号。 | <Stmt>,<ConstDecl>及<VarDecl>中的';’ |
| 缺少右小括号’)’ | j | 报错行号为右小括号**前一个非终结符**所在行号。 | 函数调用(<UnaryExp>)、函数定义(<FuncDef>)及<Stmt>中的')’ |
| 缺少右中括号’]’ | k | 报错行号为右中括号**前一个非终结符**所在行号。 | 数组定义(<ConstDef>,<VarDef>,<FuncFParam>)和使用(<LVal>)中的']’ |
| printf中格式字符与表达式个数不匹配 | l | 报错行号为**‘printf’**所在行号。 | Stmt →‘printf’‘(’FormatString{,Exp}’)’‘;’ |
| 在非循环块中使用break和continue语句 | m | 报错行号为**‘break’与’continue’**所在行号。 | <Stmt>→‘break’‘;’|‘continue’‘;’ |

（3）**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。

（4）其他类型的错误，错误的行号以能够断定发现出错的第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。

为方便对照, 下文给出了文法符号与可能存在的错误的对应关系:

编译单元    CompUnit → {Decl} {FuncDef} MainFuncDef

声明  Decl → ConstDecl | VarDecl

常量声明    ConstDecl → 'const' BType ConstDef { ',' ConstDef } ';' // i

基本类型    BType → 'int'

常数定义    ConstDef → Ident { '[' ConstExp ']' } '=' ConstInitVal  // b k

常量初值    ConstInitVal → ConstExp

    | '{' [ ConstInitVal { ',' ConstInitVal } ] '}'

变量声明    VarDecl → BType VarDef { ',' VarDef } ';' // i

变量定义    VarDef → Ident { '[' ConstExp ']' } // b

    | Ident { '[' ConstExp ']' } '=' InitVal // k

变量初值    InitVal → Exp | '{' [ InitVal { ',' InitVal } ] '}'

函数定义    FuncDef → FuncType Ident '(' [FuncFParams] ')' Block // b g j

主函数定义   MainFuncDef → 'int' 'main' '(' ')' Block // b g j

函数类型    FuncType → 'void' | 'int'

函数形参表   FuncFParams → FuncFParam { ',' FuncFParam }

函数形参    FuncFParam → BType Ident ['[' ']' { '[' ConstExp ']' }]  //   b k

语句块     Block → '{' { BlockItem } '}'

语句块项    BlockItem → Decl | Stmt

语句  Stmt → LVal '=' Exp ';' | [Exp] ';' | Block // h i

    | 'if' '(' Cond ')' Stmt [ 'else' Stmt ] // j

    | 'while' '(' Cond ')' Stmt // j

    | 'break' ';' | 'continue' ';' // i m

    | 'return' [Exp] ';' // f i

    | LVal '=' 'getint''('')'';' // h i j

    | 'printf''('FormatString{,Exp}')'';' // i j l

表达式 Exp → AddExp 注：SysY 表达式是int 型表达式

条件表达式   Cond → LOrExp

左值表达式   LVal → Ident {'[' Exp ']'} // c k

基本表达式   PrimaryExp → '(' Exp ')' | LVal | Number

数值  Number → IntConst

一元表达式   UnaryExp → PrimaryExp | Ident '(' [FuncRParams] ')' // c d e j

        | UnaryOp UnaryExp

单目运算符   UnaryOp → '+' | '−' | '!' 注：'!'仅出现在条件表达式中

函数实参表   FuncRParams → Exp { ',' Exp }

乘除模表达式  MulExp → UnaryExp | MulExp ('\*' | '/' | '%') UnaryExp

加减表达式   AddExp → MulExp | AddExp ('+' | '−') MulExp

关系表达式   RelExp → AddExp | RelExp ('<' | '>' | '<=' | '>=') AddExp

相等性表达式  EqExp → RelExp | EqExp ('==' | '!=') RelExp

逻辑与表达式  LAndExp → EqExp | LAndExp '&&' EqExp

逻辑或表达式  LOrExp → LAndExp | LOrExp '||' LAndExp

常量表达式   ConstExp → AddExp 注：使用的Ident 必须是常量

格式字符串:

<FormatString> → '"'{<Char>}'"' // a

【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。

【样例输入】

const int const1 = 1, const2 = -100;

int change1;

int gets1(int var1,int var2){

   const1 = 999;

   change1 = var1 + var2          return (change1);

}

int main(){

   change1 = 10;

   printf("Hello World$");

   return 0;

}

【样例输出】

4 h

5 i

9 a

【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，有助于完善编译器的设计、开发与调试。

（4）**每一行中最多只有一个错误。**

**（5）本次作业需要建立符号表。**

【参考资料】教材 第18章 18.3.1 18.3.6

                   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。

【开发语言及环境】用 C/C++/JAVA 实现，机房安装的 C/C++ 开发环境是 CLion 2018.3.4 和 CodeBlocks 20.03；Java 的开发环境为 IDEA 2018.3.6 社区版。产生的 MIPS 汇编在 Mars 模拟器（更正版）上运行，平台支持 C++11 标准。评测机所采用的编译学生代码的版本是：C/C++ gcc 8.1.0，Java jdk 1.8。

【辅助工具  2021实验作业调试（见在线实验）

【文档要求】完成 2021 错误处理阶段设计文档 （可在语法分析阶段设计文档基础上扩充完成）

【提交形式】将所开发的编译器程序的源文件（.cpp/.c/.h/.java，不含工程文件）打包为zip或rar后提交。对于使用 java 开发的编译器，程序运行的入口为 src 目录下 Compiler.java 中的 main 方法。上传请直接打包 src 文件夹，如果引用了第三方外部包（不推荐），请将外部 jar 包文件放到 bin 目录下，bin 和 src 两个文件夹同级，将 bin 和 src 一起打包后提交即可。注意 mac 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。