错误处理实验设计报告

——18373528杨凌华

一、新增数据结构

1）符号表

对于每一个符号表项，建立一个结构体：

```
struct Symbol
{
    string idenfr; // 标识符
    int kind;
    int type;
    int level; // 作用域的层数
    int dim; // 维数
};

enum KindEnum { // kind的枚举
    CONST, // 常量
    VAR, // 变量
    FUNC, // 函数
    PARAM // 参数
};

enum TypeEnum { // type的枚举
    CHAR,
    INT,
    VOID
};
```

符号表则为 Symbol 的数组 `Symbol symTab[10000];`，采用栈式结构

同时为符号表建立一个 [子程序索引表] 记录每一层域的索引 `int subProcIndexTable[1000];`

其中特殊的，对于函数，其函数名属于外层域，函数参数以及函数体里定义的量属于内层域，当出函数体之后，将函数体里定义的变量全部从符号表中弹栈，仅保留函数名和参数，其中参数符号表项的 idenfr 清空为 空串

2）错误信息打印

专门建立一个 ErrorProcessing类，为每条错误信息建立一个ErrorOutput结构体：

```
struct ErrorOutput
{
    int lineID; // 错误所在行号
    string errType; // 错误类别码
};
```

为 ErrorOutput 建立比较函数

建立一个错误信息数组 `vector<ErrorOutput> errOutVec;`，为其建立一个按行号递增排序的 `void sortErrOutVec();` 函数和一个去重插入函数 `void insertError(int lineID, string errType);`

在所有主程序执行的最后，调用 `void outputErrors()` 函数将错误信息表中的所有错误信息项按行号递增依次输出到 error.txt 文件中

二、错误解析与处理

a、**【非法符号或不符合词法】**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

相关文法定义：

- <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} " 且 非空

方法：在**词法分析**阶段判断

b、**【名字重定义】**同一个作用域内出现相同的名字（**不区分大小写**）

相关文法定义：

- <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>}
- <声明头部> ::= int<标识符> |char<标识符>
- <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>}'}'}'}'
- <无返回值函数定义> ::= void<标识符>'('<参数表>')''{'<复合语句>'}'
- <参数表> ::= <类型标识符><标识符>{,<类型标识符><标识符>}| <空>

方法：查同一层的符号表，转换为小写进行比较

c、**【未定义的名字】**引用未定义的名字

相关文法定义：

- <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句>
- <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>
- <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>
- <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>')'
- <读语句> ::= scanf '('<标识符>')'

方法：检查整个符号表，转换为小写进行比较

d、**【函数参数个数不匹配】**函数调用时实参个数大于或小于形参个数

e、**【函数参数类型不匹配】**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

相关文法定义：

- < 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 >

统一处理方法：先逐个依次检查参数类型是否匹配，同时进行参数计数，当检测到类型不匹配时就将 'e' 错误信息存放到错误信息数组中，放之前先检查上一条错误信息是否与之重复，重复则不再放入；当计数完毕后再检查参数数目是否匹配，如果不匹配就将 'd' 错误信息加入到错误信息数组中，加入之前先检查上一条错误信息是否是同一行的 'e' 错误，如果是的话，就将其覆盖，意图是：当同一行同时出现类型、数目不匹配时，优先输出数目不匹配。

f、**【条件判断中出现不合法的类型】**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

相关文法定义：

- < 条件 > ::= < 表达式 > < 关系运算符 > < 表达式 >

方法：在表达式解析递归子程序中添加一个返回值type，返回该表达式的类型，然后判断 '==' 两边表达式解析递归子程序的返回值是否都为 INT。

g、**【无返回值的函数存在不匹配的return语句】**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

相关文法定义：

- < 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 >
- < 语句列 > ::= { < 语句 > }
- < 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 > '}'
- < 返回语句 > ::= return['(' < 表达式 > ')']

方法：

因为从 < 无返回值函数定义 > 到 < 返回语句 > 之间存在多层函数递归调用，因此势必要进行跨函数状态处理，有两种解决方法：一是函数传参，二是设置宏观全局状态变量。前者容易维护但是参数传递需要改动较多，后者写起来简单，不过容易忘记维护。根据个人喜好，我选择了后者，设置了

```
int funcRetFlag = -1;// -1:不在函数里 0：在无返回值函数里 1：在有返回值函数里
```

当进入void函数中时，将funcRetFlag设置为0，出函数时设置为-1

用来存储当前处理的单词是否处于函数中，如果是，又是处于哪种类型的函数中。

那么在进入解析<返回语句>的递归子程序中时，如果检测到 '(' 就判断 `funcRetFlag == 0` 是否成立，如果成立则报错

h、**【有返回值的函数缺少return语句或存在不匹配的return语句】**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

相关文法定义：

- <有返回值函数定义> ::= <声明头部>'('<参数表>')''{'<复合语句>'}'
- <复合语句> ::= [<常量说明>] [<变量说明>] <语句列>
- <语句列> ::= {<语句>}
- <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>'}'
- <返回语句> ::= return['('<表达式>')']

方法：

同样采用宏观全局状态变量，

```
int funcRetFlag = -1;// -1:不在函数里 0：在无返回值函数里 1：在有返回值函数里
int hasRet = 0; // 当前带返回值的函数 是否 有return语句
int retType = -1; // 当前带返回值函数 的 返回值类型 0: CHAR 1: INT
```

当进入有返回值的函数中时，将funcRetFlag设置为1，出函数时设置为-1

当进入返回语句解析递归子程序时，判断如果funcRetFlag == 1，则将hasRet置为1

首先检测是否有

1、return;

2、return();

3、表达式类型与返回值类型不一致

有则报错

之后检测hasRet是否为0，为0则报错

i、**【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

相关文法定义：

- <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句>
- <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>

方法：

在解析相关语法成分时，判断parseExpr()的返回值是否为CHAR，是则报错

j、**【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赋值或用scanf获取a的值，则报错。

相关文法定义：

- <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>

- < 读语句 > ::= scanf '(' < 标识符 > ')'

方法：

在相关子程序中，判断在符号表中查找相关的标识符，判断其kind值是否为 CONST，是则报错

k、【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 （7种语句末尾，for语句中，常量定义末尾，变量定义末尾）

相关文法定义：

- < 常量说明 > ::= const < 常量定义 > ;{ const < 常量定义 > ;}
- < 变量说明 > ::= < 变量定义 > ;{ < 变量定义 > ;}
- < 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 > '}'
- < 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 >

方法：

此处需要注意输出的行号，如果 ' ; ' 在行末，如果缺少，则会读到下一行，此时输出的行号就会变成出错的行号的下一行。

解决方式：为每一个单词添加一个bool属性isAtHead，如果为真，则表示该单词在行首

那么当检测到缺少分号时，就判断当前读到的单词的isAtHead是否为真，为真则输出 当前行号 - 1 ，否则就输出当前行号

l、【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（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）

相关文法定义：

- < 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 > '}'
- < 主函数 > ::= void main‘(’)' ‘{’ < 复合语句 > ‘}’
- < 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句 >
- < 条件语句 > ::= if '(' < 条件 > ')' < 语句 > [else < 语句 >]
- < 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 >
- < 情况语句 > ::= switch ‘(’ < 表达式 > ')' ‘{’ < 情况表 > < 缺省 > ‘}’
- < 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 > ')'
- < 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 > ')']

方法：

此处需要注意输出的行号，如果 ') ' 在行末，如果缺少，则会读到下一行，此时输出的行号就会变成出错的行号的下一行。

解决方式：为每一个单词添加一个bool属性isAtHead，如果为真，则表示该单词在行首

那么当检测到缺少 ')' 时，就判断当前读到的单词的isAtHead是否为真，为真则输出 当前行号 - 1 ，否则就输出当前行号

m、【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（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）

相关文法定义：

- < 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 > }'}'}'}'
- < 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句 >
- < 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 >

方法：

此处需要注意输出的行号，如果 '] ' 在行末，如果缺少，则会读到下一行，此时输出的行号就会变成出错的行号的下一行。

解决方式：为每一个单词添加一个bool属性isAtHead，如果为真，则表示该单词在行首

那么当检测到缺少 ']' 时，就判断当前读到的单词的isAtHead是否为真，为真则输出 当前行号 - 1 ，否则就输出当前行号

n、【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}}

相关文法定义：

- < 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 > }'}'}'}'

方法：

在解析<变量定义及初始化>时，记录中括号中的无符号整数，然后分一维、二维，建立循环遍历 '=' 右边的成分，出现异常即报错

o、【<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。int x='c';int y;switch(y){case('1') ....}

相关文法定义：

- < 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 > }'}'}'}'
- < 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 >

方法：

对于<常量>解析递归子程序parseConst()，返回值为常量的类型，INT和CHAR

对于switch语句，采用传递type参数进入情况子语句进行类型匹配检查

p、【缺少缺省语句】switch语句中，缺少<缺省>语句。

相关文法定义：

- < 情况语句 > ::= switch '(' < 表达式 > ')' '{' < 情况表 > < 缺省 > '}'
- < 缺省 > ::= default : < 语句 >

方法：直接在出了<情况表>子程序之后，看有没有"DEFAULTTK"

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 (不区分大小写)
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。

单词名称	类别码	单词名称	类别码	单词名称	类别码	单词名称	类别码
标识符	IDENFR	else	ELSETK	-	MINU	=	ASSIGN
整形常量	INTCON	switch	SWITCHTK	*	MULT	;	SEMICN
字符常量	CHARCON	case	CASETK	/	DIV	,	COMMA
字符串	STRCON	default	DEFAULTTK	<	LSS	(	LPARENT
const	CONSTTK	while	WHILETK	<=	LEQ	)	RPARENT
int	INTTK	for	FORTK	>	GRE	[	LBRACK
char	CHARTK	scanf	SCANFTK	>=	GEQ	]	RBRACK
void	VOIDTK	printf	PRINTFTK	==	EQL	{	LBRACE
main	MAINTK	return	RETURNTK	!=	NEQ	}	RBRACE
if	IFTK	+	PLUS	:	COLON		