**《编译原理课程设计》报告**

**MiniC编译器设计与实现**

分组序号： 1

设计地点： 微202

电子邮件： 1933196334@qq.com

分组成绩：

任课教师： 李村合

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 专业班级 | 计算(实验) 1901班 | | | | | |
| 姓名 | 张怀治 | 刘文杰 | 周志伟 | 汤佳蕾 | 韦欣潼 | 周小钊 |
| 学号 | 1909070305 | 1909070204 | 1909070306 | 1909070122 | 1909070225 | 1909070224 |
| 成绩比例(%) | 16.67% | 16.67% | 16.67% | 16.67% | 16.67% | 16.67% |
| 成绩 |  |  |  |  |  |  |

2022 年 4 月13 日

**目录**

[1 课程设计目的 4](#_Toc100689862)

[2 课程设计内容 4](#_Toc100689863)

[3 课程设计原理 4](#_Toc100689864)

[4 系统需求分析 5](#_Toc100689865)

[4.1 MiniC编译器总体介绍 5](#_Toc100689866)

[4.2 功能需求 5](#_Toc100689867)

[4.3 用到的主要关键词 6](#_Toc100689868)

[4.4 MiniC语言的文法 6](#_Toc100689869)

[5 系统设计与实现 7](#_Toc100689870)

[5.1 编译程序主要功能模块 7](#_Toc100689871)

[5.2 词法分析子程序 8](#_Toc100689872)

[5.2.1 单词分类 8](#_Toc100689873)

[5.2.2 具体设计思路 8](#_Toc100689874)

[5.3 语法分析子程序 11](#_Toc100689875)

[5.3.1 具体设计思路 11](#_Toc100689876)

[5.4 错误处理子程序 13](#_Toc100689877)

[5.4.1 错误类型及解释 13](#_Toc100689878)

[5.4.2 符号表的建立 14](#_Toc100689879)

[5.4.3 错误处理方法 16](#_Toc100689880)

[5.5 代码生成程序 17](#_Toc100689881)

[5.5.1 生成中间代码 17](#_Toc100689882)

[5.5.2 生成mips代码 18](#_Toc100689883)

[5.6 代码优化 19](#_Toc100689884)

[5.6.1 常数优化 19](#_Toc100689885)

[5.6.2 内联函数优化 19](#_Toc100689886)

[5.6.3 临时寄存器优化 20](#_Toc100689887)

[5.6.4 全局寄存器优化 20](#_Toc100689888)

[5.6.5 其他优化 20](#_Toc100689889)

[5.7 界面设计子程序 21](#_Toc100689890)

[6 系统测试与运行结果分析 21](#_Toc100689891)

[6.1 二维数组测试程序 21](#_Toc100689892)

[6.2 汉诺塔测试程序 23](#_Toc100689893)

[6.3 累加测试程序 26](#_Toc100689894)

[6.4 错误处理测试程序 28](#_Toc100689895)

[7 心得体会 29](#_Toc100689896)

# 1 课程设计目的

1、根据所掌握的编译原理课程的基本知识，编写出一个 MiniC 编译器；

2、增强阅读和编写程序的能力；

3、理解词法分析、语法分析和语义分析在编译程序中的作用；

4、掌握词法分析、语法分析和语义分析程序的实现方法；

5、实现 MiniC 编译器的编辑、编译、连接和运行。

# 2 课程设计内容

用 MFC 编写一个可视化的 MiniC 编译器，要求界面美观，使用方便，能够实现自定义 MiniC语言的编译。

支持的语句和运算：

1、数据类型：int、char、void、const

2、语句：赋值、if-else、switch-case、default、while、for、scanf、printf、return

3、算术运算：+、－， 、\*、/

4、关系运算：==、>、<、>=、<=、!=

5、其它符号：（ ）、{ }、[ ]、:、;

6、支持函数的定义、调用

7、支持复合语句，即｛｝包含的语句

8、注释：C类型的多行注释/\* \*/

# 3 课程设计原理

MiniC语言可以看成C语言的子集，它的编译程序是一个编译执行环境。

MiniC的编译程序和目标程序的执行程序都是用C++语言编写的，因此MiniC语言可在配备C语言的任何机器上实现。它的编译过程采用从左到右逐个字符地读入源程序，对构成源程序的字符流进行扫描和分解，识别出一个个单词。然后根据预先定义的文法规则，采用递归下降子程序的文法分析方法从源程序单词符号串中识别出语法成分，并进行语法检查。接下来根据语义分析审查源程序有无语义错误，为代码生成阶段收集类型信息。当以上过程都顺利无误时，编译器会生成中间代码作为内部表示形式，最终生成目标代码，也就是Mips汇编代码，最后在Mips模拟器上运行。

# 4 系统需求分析

MiniC语言编译过程采用从左到右扫描输入符号串，进行最左推导，再向右看一个符号的方式，对源程序分别进行词法分析、语法分析、语义分析、中间代码生成、目标代码生成，此外，我们还设计了错误处理功能。用错误处理程序对词法和语法分析研究遇到的错误给出在源程序中出错的位置和错误性质，进行错误局部化处理，并输出错误信息。MiniC编译器的工作过程如下所示：

*输入源代码→词法分析→语法分析→错误处理→中间代码生成→汇编代码生成*

## 4.1 MiniC编译器总体介绍

主要将编译器分为前后端两个部分，前者包括界面的搭建、交互功能的实现以及程序运行结果的展示，后端包括词法分析、语法分析、错误处理、中间代码生成及优化，前后端以打包好的编译器后端的可执行文件为分界，在一定程度上相互独立。后文将按照前后端的各个模块依次进行介绍大致实现流程和重难点。

## 4.2 功能需求

1、用C++语言实现了类C语言，叫做MiniC语言；

2、使用LL(1)文法，从左到右扫描输入符号串，进行最左推导，再向右看一个符号；采用自顶向下的递归子程序法实现语法分析，并用C++语言实现了词法分析器、语法分析器、汇编代码生成器，能直接生成mips汇编代码。

3、在声明中实现了对静态常量、变量和函数声明的支持；赋值语句的=，基本的数学运算+、-、\*、/ ；

4、使用函数实现了对过程的调用；

5、使用{ }实现复合语句；

6、使用函数scanf()来输入数据，使用printf()来同时输出一个或多个数据；

7、在循环分支语句中实现if语句，if…else…语句，for(…,…,…)…语句，while…语句，switch…case…语句；

8、关系表达式包括==(等于)、!=(不等于)、<、<=、>、>=；

9、用VS2019的Microsoft基础类库MFC进行界面设计。

## 4.3 用到的主要关键词

1、关键字:

IDENFR、INTCON、CHARCON、STRCON、CONSTTK、INTTK、CHARTK、VOIDTK、MAINTK、IFTK、LSETK、SWITCHTK、CASETK、DEFAULTTK、WHILETK、FORTK、SCANFTK、PRINTTK、RETURNTK

2、操作符:

PLUS、MINU、MULT、DIV、LSS、GRE、ASSIGN

3、界符:

COLON、SEMICN、COMMA、LPARENT、RPARENT、LBRACK、RBRACK

4、复杂操作符:

EQL(==)、NEQ(!= )、LEQ (<=)、GEQ(>=)

## 4.4 MiniC语言的文法

＜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＞}

＜整数＞ ::= ［＋｜－］＜无符号整数＞

＜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＞

＜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＞'}'

＜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＞')']

# 5 系统设计与实现

## 5.1 编译程序主要功能模块

1、功能一：预处理掉相关注释，函数为：Pretreat:: pretreat();

2、功能二：是否出现非法字符，函数为：Lexer:: isString(char c)；

3、功能三：获取下一个token，函数为：Lexer:: getsym()；

4、功能四：根据文法规则调用递归子程序，函数为：Parser:: getProgram()；

5、功能五：根据错误类型判断错误位置，函数为：Error:: error(char errorCode)；

6、功能六：生成中间代码，函数为：MidCode:: insertMidCode()；

7、功能七：生成mips汇编代码，函数为：MipsCode:: generateMipsCodes()；

## 5.2 词法分析子程序

### 5.2.1 单词分类

对于词法分析，我们的大致思路是对单词种类进行分类，对保留字和分界符采用一符一类的方法，之后根据状态转移图进行不同种类单词的识别。简单来说，就是根据词法规则分析出状态转移图，然后根据状态转移图进一步写出词法分析程序。

词法分析是整个编译器的第一步，也是较为简单的一个环节。我们将单词分为以下五类。

1、以<字母>开始的，包括标识符、const、int……return等；

2、以<数字>开始的，主要是整形常量 ；

3、以单引号开始的字符常量 ；

4、以双引号开始的字符串 ；

5、以特殊字符开始的特殊符号；

编译器所能识别的单词和相应的类别码见表1：

**表1 单词和相应的类别码**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **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 |
| 整型常量 | 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 | { | LBRACK |
| Main | MAINTK | return | RETURNTK | != | NEQ | } | RBRACK |
| if | IFTK | + | PLUSE | : | COLON |  |  |

### 5.2.2 具体设计思路

1、预处理

在预处理阶段，创建了ifstream和ofstream分别对应输入文件和输出文件，同时对CODEN和CODES数组进行了初始化，为后续的处理做了准备。在明确了单词的类别码之后，我对类别码分配了内部编码，并将类别码保存在了一个string数组中。以下是我们的保存类别码的数组的初始化函数。该数组的下标对应的就是该类单词的内部编码。

void init\_CODEN() {

CODEN[0] = "IDENFR"; // 标识符

CODEN[1] = "INTCON"; // 整型常量

CODEN[2] = "CHARCON"; // 字符常量

CODEN[3] = "STRCON"; // 字符串

CODEN[4] = "CONSTTK";

CODEN[5] = "INTTK";

CODEN[6] = "CHARTK";

CODEN[7] = "VOIDTK";

CODEN[8] = "MAINTK";

CODEN[9] = "IFTK";

CODEN[10] = "ELSETK";

CODEN[11] = "SWITCHTK";

CODEN[12] = "CASETK";

CODEN[13] = "DEFAULTTK";

CODEN[14] = "WHILETK";

CODEN[15] = "FORTK";

CODEN[16] = "SCANFTK";

CODEN[17] = "PRINTFTK";

CODEN[18] = "RETURNTK";

CODEN[19] = "PLUS";

CODEN[20] = "MINU";

CODEN[21] = "MULT";

CODEN[22] = "DIV";

CODEN[23] = "LSS";

CODEN[24] = "LEQ";

CODEN[25] = "GRE";

CODEN[26] = "GEQ";

CODEN[27] = "EQL";

CODEN[28] = "NEQ";

CODEN[29] = "COLON";

CODEN[30] = "ASSIGN";

CODEN[31] = "SEMICN";

CODEN[32] = "COMMA";

CODEN[33] = "LPARENT";

CODEN[34] = "RPARENT";

CODEN[35] = "LBRACK";

CODEN[36] = "RBRACK";

CODEN[37] = "LBRACE";

CODEN[38] = "RBRACE";

}

2、从文件中读取到缓冲区

我们使用了C++的<fstream>来进行文件的读取。通过循环不断将文件中的一行代码传入到程序的输入缓冲区，也就是buffer数组中，然后调用getsym()函数不断获取单词并输出到输出文件直到该行的每一个字符都已经被扫描过。然后再读取文件的下一行代码进行同样的处理。以下是这一部分相关的代码：

while (!ifile.eof()) {

// 初始化相关变量

cnt = 0;

cnt\_line++;

// 从文件中读取一行字符串

ifile.getline(buffer, 256);

cnt\_len = strlen(buffer);

// 对该行字符串进行分析

while (cnt < cnt\_len) {

getsym();

outputsym();

}}

3、从缓冲区获取单词

这是本次作业最重要的部分，主要涉及的是getsym()函数，目标就是从输入缓冲区中获取下一个单词。我们设置了三个跟读取指针相关的函数，用于从字符缓冲区获取下一个字符。

（1）int getnbc()：从字符缓冲区获取下一个非空字符；

（2）void getch()：从字符缓冲区获取一个字符；

（3）void ungetch()：从字符缓冲区回退一个字符；

4、输出获取到的单词

我们将getsym()函数的返回值设置为了内部编码，也就是每个单词名称对应的CODEN数组下标。在主函数中，通过收到的返回值按照不同的单词种类进行输出。单词的输出也分两种情况，倘若是特殊字符，则输出CODES数组中保存的对应的名称；其余情况则输出TOKEN变量中保存的字符串。这么做的主要原因是部分的保留字会出现大写，例如本次作业的样例中的const，这导致不能通过内部编码直接输出预设的字符串，而必须输出程序运行过程中实际读取到的字符串。

## 5.3 语法分析子程序

语法分析的主要任务是根据文法规则，从源程序单词符号串中识别出语法成分，并进行语法检查。在本次编译器设计中，我们的程序使用的是自顶向下的递归子程序法。具体来说，就是给语法的每一个非终结符都编写一个分析程序，当根据文法和当前的输入符号预测到要用到某个非终结符去匹配输入串时，就调用该非终结符的分析程序。本次设计的文法规则中没有左递归的文法，但是部分非终结符的规则右部的多个选择会有一些相交的首符号集合。因此，遇到这些非终结符时我们使用了超前扫描的方法来规避回溯的操作。关于词法分析程序和语法分析程序的关系，我们采用了词法分析程序作为单独的子程序的实现方案。在分析语法成分的同时不断从词法分析子程序中取单词，从而提高程序的效率。

### 5.3.1 具体设计思路

1、预处理

打开相应的输入流文件和输出流文件，然后初始化词法分析的内部编码。

2、预读取一个单词并开始getProgram()函数

在我们的文法规则中，根节点是<程序>这个语法成分，因此我们的所有语法分析都是从getProgram()这个函数开始的。在预读取一个单词之后，我们可以在<程序>的右部中寻找可能成立的选择，并根据不同选择的前部和预读取的单词类别进行匹配，进而调用递归子程序进行下一步的扫描。以下是我们的主函数：

int main() {

openFiles(); // 打开所有文件

getsym(); // 词法分析

getProgram(); // 语法分析

outputMidCodes(); // 输出中间代码

splitBlock(); // 分割代码块

generateMipsCodes(); // 生成mips代码

outputMipsCodes(); // 输出mips代码

closeFiles(); // 关闭所有文件

return 0;}

3、不断调用递归子程序

在调用了getProgram()函数后，我们要做的就是不断地根据文法规则调用递归子程序，直至词法分析得到的所有单词都能根据文法规则找到对应的语法成分为止。

本次作业虽然在思路上并不复杂，然而，主要难度在于众多的语法成分以及部分语法成分的具有相同首符号的右部。前者需要我们注意细节从而减少错误的发生，后者则需要我们使用超前扫描的方法，确定具体的语法成分。

例如对于<程序>右部的<变量说明>和<有返回值函数定义>这两个的语法成分，具有相同的首符号，均为int或者char，同时第二个符号也都是标识符。因此，我们需要至少超前扫描两个单词才能具体确认右部匹配的语法成分。就这个问题而言，当第三个语法成分为左括号时代表的是<有返回值函数定义>，反之则是<变量说明>。以下是关于这个部分的代码：

while (symbol == INTTK || symbol == CHARTK) {

trysym(); // 超前扫描一个单词

trysym(); // 超前扫描一个单词

if (symbol != LPARENT) {

hasVariable++;

ungetsym();

outputlastsym();

getVariableDeclarationForProgram();

}

else {

ungetsym(); // 回退一个单词

ungetsym(); // 回退一个单词

break;

}}

在根据每个语法成分都编写了递归子程序之后，我们就可以从<程序>这个根结点出发将语法分析子程序扫描到的单词与文法定义中的非终结符一一匹配，从而完成语法分析的整个过程。

## 5.4 错误处理子程序

错误处理的主要任务是根据给定的文法设计实现错误处理程序，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。完成错误处理主要分为两个步骤：第一个是建立符号表，第二个是根据不同类型错误可能出现的位置判断不同的错误类型。错误处理阶段使用的主要变量和函数如下：

// 以下为外部引用的变量和函数

extern ofstream errorFile;

extern int cnt\_line;

extern int lastCountLine;

// 以下为内部定义的变量和函数

void error(char errorCode);

### 5.4.1 错误类型及解释

错误类型及解释见表2，

**表2 错误类型及解释**

|  |  |  |
| --- | --- | --- |
| **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，缺少语句 |

### 5.4.2 符号表的建立

我们将符号表的建立与更新主要分为了两步：一种是添加整个层次，另一种是在当前层次下添加符号。需要向符号表中增加层次主要有以下四种函数，我使用了函数名来作为该层次的key值。

1、<程序>，暂时使用Program作为该层次的名称；

2、<有返回值函数>，使用函数名的标识符作为该层次的名称；

3、<无返回值函数>，使用函数名的标识符作为该层次的名称；

4、<主函数>，暂时使用Main作为该层次的名称；

需要在同一层次向符号表中添加项的主要有如下情况。

1、常量定义（kind=0）：相对变量定义较为简单，只有整数和字符两种情况，不会有数组或者字符串。

（1）<程序>中的<常量说明>

（2）<有返回值函数>和<无返回值函数>中的<复合语句>中的<常量说明>

2、变量定义（kind=1）

（1）<程序>中的<变量说明>

（2）<有返回值函数>和<无返回值函数>中的<复合语句>中的<常量说明>

3、有返回值函数定义（kind=2）：<程序>中的<有返回值函数>

// 将该函数名添加到符号表

initCurrentSymbol();

curSym.kind = 2;

curSym.type = 0;

curSym.name = lastIdentifier;

insertCurrentSymbol();

4、无返回值函数定义（kind=3）：<程序>中的<无返回值函数>

// 将该函数名添加到符号表

initCurrentSymbol();

curSym.kind = 3;

curSym.type = 0;

curSym.name = lastIdentifier;

insertCurrentSymbol();

5、参数表（kind=4）：<有返回值函数>和<无返回值函数>中的<参数表>

// 将参数添加到符号表中，仅有kind和type，没有value

initCurrentSymbol();

curSym.kind = 4;

if (lastTypeIdentifier == INTTK) {

curSym.type = 1;

}

else {

curSym.type = 4;

}

curSym.name = lastIdentifier;

insertCurrentSymbol();

6、主函数定义（kind=5）：

// 将main添加到符号表

initCurrentSymbol();

curSym.kind = 5;

curSym.type = 0;

curSym.name = MAIN;

insertCurrentSymbol();

### 5.4.3 错误处理方法

1、错误类型a

错误类型a是本次作业中唯一的文法错误，所以需要在文法分析阶段进行处理。主要分为以下两个情况。

（1）在有单引号包围的字符情况下，检查读取到的字符是否有非法字符。

（2）在有双引号包围的字符串情况下，检查读取到的字符是否有非法字符，并且在读取到右双引号之后检查读取到的字符串是否为空集。

2、错误类型bc

要判断错误bc的前提是要建立符号表。由于根据我们的文法，我们并不会有太多的函数嵌套，常量定义和变量定义也只有在<程序>、<有返回值函数定义>、<无返回值函数定义>中才会出现。因此在查询符号表的过程中，只需要查询当前所在函数的符号表和最外层的符号表（全局变量）即可。

所以我们在常量定义和变量定义的过程中向符号表中增加符号，在使用到标识符的地方查询符号表是否有相应的符号。需要注意的是，在建立符号表时，如果是变量及初始化，每行仅能有一个；若变量无初始化，每行可以有多个。

3、错误类型de

在函数定义的<参数表>部分，我们可以更新该函数的参数表。在函数调用<值参数表>部分，我们可以将实际的输入参数和参数表中的需要的参数进行比对，进而识别并输出错误。

4、错误类型f

该错误可能发生的位置较为单一，在条件判断语句中判断等号两边的表达式是否为整型，进而识别并输出错误。

5、错误类型gh

要判断函数的返回类型是否和定义相同，我们需要建立一个函数表。

当我们读取到<有返回值函数定义>的<声明头部>以及<无返回值函数定义>的void<标识符>部分，我们就可以判断出该函数的返回值类型和函数名称。因此我们在这里将函数名称和返回值类型加入到函数表中，当读取到<语句列>中的<返回语句>时与函数表进行比较判断，进而识别并输出错误。

6、错误类型i

该错误只会出现在<因子>和<赋值语句>中。在这些语句可能使用到数组的地方，如果接收到的数组下标不是整型，则识别并输出该型错误。

7、错误类型klm

该错误相对来说比较容易识别。在原来的语法分析的基础上，需要接收分号、右小括号和右中括号的地方，如果未能成功接收该符号，则捕捉并输出该错误。

需要注意的是，如果缺少的符号位于一行的最后一个，那么在输出错误的时候要注意行数需要定位到上一个单词所在的行数。以缺少分号为例，调用getsym()函数会获取到下一行的第一个单词，相应的行数计数器也会增加，但是我们缺少分号的错误行数应该是上一行。

8、错误类型o

该错误主要出现在变量定义及初始化和switch语句中的<常量>，需要判断等号两边的类型是否契合，否则捕捉o型错误。

9、错误类型p

该错误需要在<情况语句>中没能成功接受DEFAULTTK的symbol时捕捉。

## 5.5 代码生成程序

### 5.5.1 生成中间代码

在这一阶段，我们要做的事主要采用自顶向下的语法制导翻译技术，进行语义分析并生成中间代码。其中涉及<常量说明>、<变量说明>、<读语句>、<写语句>、<赋值语句>五种成分，我们首先对这些元素进行一定分析：

1、<常量说明>：将符号保存在符号表中，并且保存相应的常量值。

2、<变量说明>：将符号保存在符号表中，特别的对于有初始值的变量的操作类似于常量，只不过还可以进行赋值等操作。

3、<读语句>：根据读语句中的标识符，通过mips指令中的系统调用读取相关数据并保存到对应的内存地址。

4、<写语句>：根据写语句中需要数据的字符串和标识符，通过mips指令中的系统调用输出相关数据。

5、<赋值语句>：取出操作数的值，并保存到结果的内存地址即可。

除了这些成分，中间代码生成过程中难度较大的部分是关于的中间代码生成过程。 在这里我们主要使用了符号栈的思想，当读取到运算符时，弹出并检查当前在栈顶的两个元素，如果两者都是符号（常量或者变量），则生成一个新的中间临时变量作为操作结果并添加到符号表中；倘若两者中的左操作数为中间临时变量，则将运算后的结果直接保存到该中间临时变量即可。一直这样操作下去，最后将符号栈中剩余的最后一个元素，就是保存了该表达式结果的标识符。

由此，我们又设计了如下五种成分的语义分析：

1、数组相关：对于数组的取用，主要难度在于mips代码中对数组元素地址的计算，需要在数组地址的基础上加入元素的偏移。

2、函数相关：函数相对而言是较为容易实现的，因为main本身就是一个特殊的函数，我们只需要在函数调用语句的部分加入跳转指令，并在<返回语句>中跳转回跳转前的地址即可。

3、<条件语句>：首先将条件语句中的条件转化为一个跳转指令，同时对if和else分别生成两个标签ifLabel和elseLabel，前者用于条件不符合时跳转至else语句或者跳出该条件语句，后者则用于条件符合时跳出else语句。

4、<循环语句>：循环语句包含while和for两种情况。

（1）while语句：在循环开始处进行条件判断是否需要跳出循环，在循环语句最后插入跳转指令，返回while语句的条件判断处。这里使用了两个标签beginLabel和endLabel，分别用于跳转回循环开始处，和条件不满足时跳出循环语句。

（2）for语句：for语句的大致结构与while语句相似，主要是在for语句开始前增加了对循环变量的初始化操作，以及循环语句的最后需要单独增加修改循环变量的语句。与while语句相同的是，这里使用了两个标签beginLabel和endLabel，分别用于跳转回循环开始处，和条件不满足时跳出循环语句

5、<情况语句>：对于情况语句，我们采用了比较简单的办法，就是将每个case转化为了一个if语句进行分析，如果case语句中的变量和主变量不同则跳过该case语句，在每个case语句的最后增加一个跳转出整个switch语句的指令，从而完成了整个情况语句的代码构建。

### 5.5.2 生成mips代码

在我们的编译器设计中，代码逻辑上的跳转等已经在生成中间代码的过程中完成了设计，所以生成mips指令过程的主要工作就是为中间代码寻找对应的mips指令进行翻译即可。

从中间代码到mips代码，最大的变化就是运算的操作数从无限的变量到有限的寄存器中。因此我们在进行相应的运算之前需要将符号表或者内存地址中的值取出到寄存器中，并在运算结束后保存到内存地址中。在本次设计中，我们没有对寄存器的使用进行优化，而是采用了每条语句后还原的思路，保证每条语句之间的寄存器使用互不影响。

从上述描述中，不难发现，生成mips指令阶段的主要任务就是在匹配相应的mips指令的基础上分配内存地址和寄存器。我们在这里主要分析函数调用和数组取用两个过程进行分析：

1、函数调用：函数调用的过程首先将相关变量（包括函数参数、变量和生成中间代码过程中产生的临时变量）、$ra和$fp寄存器保存到栈中，之后就可以修改栈指针并跳转至相应的标签。

2、数组元素：数组元素的取用主要是要进行地址的计算，这里以获取二维数组中的元素为例，先将两个下标分别保存至$t0和$t1寄存器，之后根据$t0 \* column($t3) + $t1计算出该数组元素相对于数组地址的偏移，进而找出该数组元素在内存地址中位置。

## 5.6 代码优化

### 5.6.1 常数优化

在代码生成的过程中，如果标识符对应的是常数，则在大多数的情况下可以进行一定的优化。

1、加减法运算中，遇到一个操作数为常数时，可以用addi和subi；

2、乘除法运算中，遇到操作数中有0或1的情况可以进行简化，例如x\*1、x\*0、x/1、0/x等情况下可以不进行调用乘除操作就得到相应的运算结果；

3、条件跳转语句中,如条件两边均为常数,可以直接得出结果以判断是否跳转；

4、运算过程中可以使用$0寄存器替代数字0；

5、数组调用以及数组赋值过程中，若数组下标为常数，可以直接计算出正确的地址。

通过用常数替换从内存中获取数据并用寄存器参与运算的过程，可以节省大量的load操作的时间。

### 5.6.2 内联函数优化

对于没有跳转语句、函数调用语句、数组引用的函数，可以使用内联函数进行优化，从而减少函数调用带来的跳转等必然消耗。通过内联函数优化，可以节省很多保存现场的时间，此外还可以减少很多参数传递操作所需要的时间。当该函数位于循环语句中，内联优化的效果会格外明显。

在进行函数内联之前，我们需要对原函数中的变量名进行预处理，如果是原函数的局部变量则需要重命名并加入到外层函数的符号表中，如果是全局变量则不需要进行处理。

### 5.6.3 临时寄存器优化

对于临时寄存器，可以建立寄存器池对临时寄存器进行统一分配。当遇到新的未分配寄存器的变量时从寄存器池中获取新的寄存器，当遇到所有的寄存器都已经被使用的情况时，则需要考虑释放寄存器。在释放寄存器的同时，要按照寄存器对应的变量是否被改写判断寄存器中的值是否需要写回内存。当一个基本块结束时，我们需要释放所有的临时寄存器。

### 5.6.4 全局寄存器优化

对于全局寄存器，我们可以通过引用计数法或者图着色算法进行全局寄存器分配。 首先我们先来介绍全局寄存器相较于临时寄存器的优点。临时寄存器的生存周期是一个基本块，当基本块结束时需要释放临时寄存器，而在下个基本块中如果需要再次使用到这个变量，则需要重新从内存中读取并写入寄存器。这种操作对于仅需使用一次的变量没什么影响，但是对于那些经常需要使用的变量，分配一个全局寄存器会比使用临时寄存器节省大量的存取内存的时间。

基于引用计数法分配全局寄存器的方法相对来说比较简单。以一个函数为一个全局域，统计该域内所有变量的引用次数，对于引用次数较多的变量则分配全局寄存器。在使用引用计数法统计变量的引用次数时，要注意调整循环变量的引用次数的权重，因为单纯的引用计数可能会减少循环变量的作用。

图着色法分配寄存器需要首先进行基本块划分与活跃变量分析。基本块的起始语句有三种情况：函数的第一条语句；跳转语句跳转到的语句；跳转语句之后的语句。活跃变量分析则是根据基本块的划分结果每个基本块的in、out、def、use集合，从而进行后续的图着色。

### 5.6.5 其他优化

以下都是一些细节之处的优化，故在此一并指出：

1、遇到#T1=#T2+#T3；a=#T1；这种情况可以简化为a=#T2+#T3；

2、删除无用代码，对于结果操作数在后文中未使用的情况下，直接删除；

3、对于循环语句，尽可能地使用beq和bne语句替换跳转语句以提高性能。

## 5.7 界面设计子程序

界面设计根据需求设计了打开、编译和查看功能，并配有输入和输出编辑框。

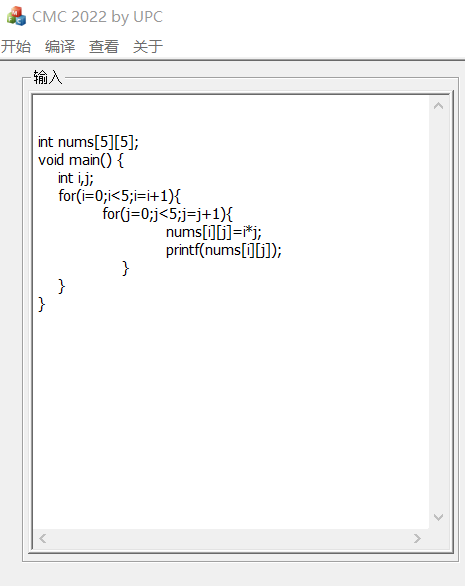
1、对于打开文件的功能：在点击“开始”后，程序运行会打开文件夹选择框，并根据用户选择的文件路径打开文件并查看文件内容，同时将文件内容显示在输入编辑框中。“打开”选项的消息处理函数为：CMFCApplication1Dlg::On32793()。

2、对于编译程序的功能：在点击“编译”后，程序会将输入框中的内容保存到testfile.txt文件中，并启动compiler.exe对程序进行编译，这个选项是界面设计子程序与编译器子程序连接的接口。“编译”选项的消息处理函数为：CMFCApplication1Dlg::On32772()。

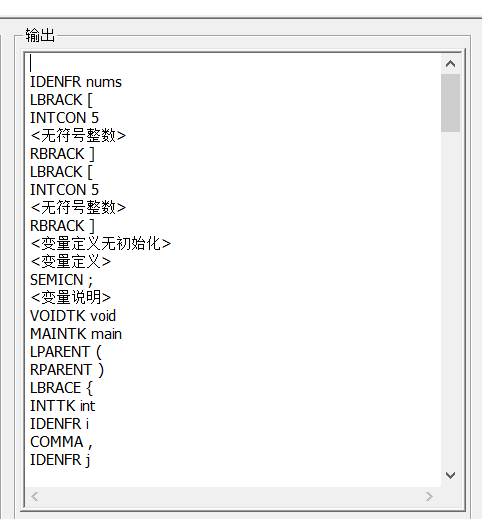
3、对于查看文件的功能：在“查看”菜单栏中，有“中间代码”、“汇编代码”、“词法分析”、“错误信息”四个选项，每个选项分别对应相应的txt文件，点击选项会打开相应的txt文件并将内容显示在输出编辑框中。四个选项对应的消息处理函数分别为：CMFCApplication1Dlg::On32776()、CMFCApplication1Dlg::On32777()、CMFCApplication1Dlg::On32778()和CMFCApplication1Dlg::On32779()。

# 6 系统测试与运行结果分析

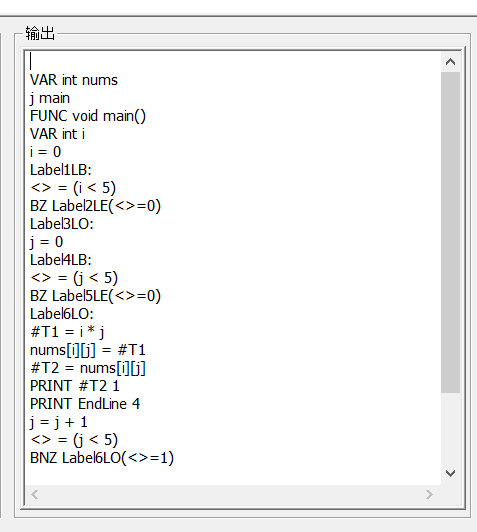
## 6.1 二维数组测试程序



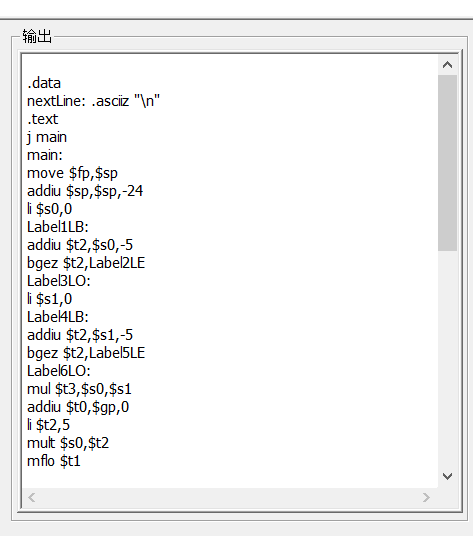
**图1 测试用例1**



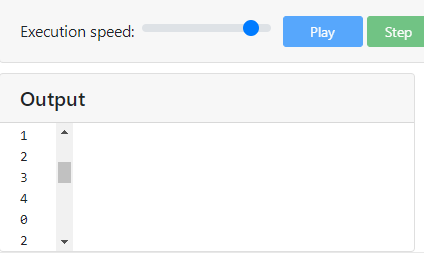
**图2 词法分析1**



**图3 中间代码1**

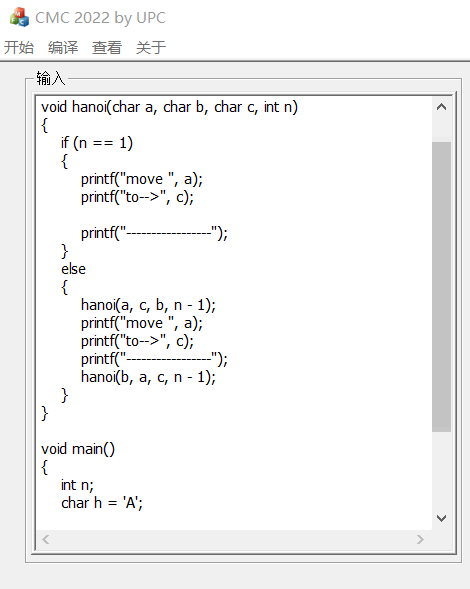


**图4 汇编代码1**

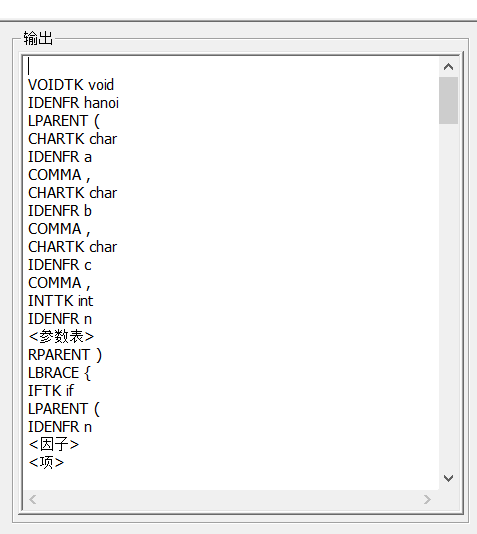


**图5 运行结果1**

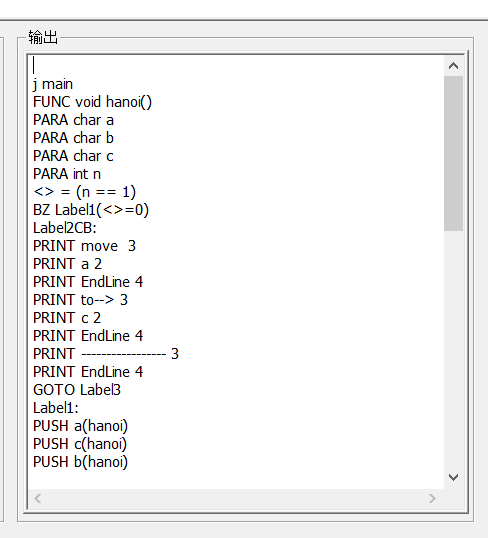
## 6.2 汉诺塔测试程序



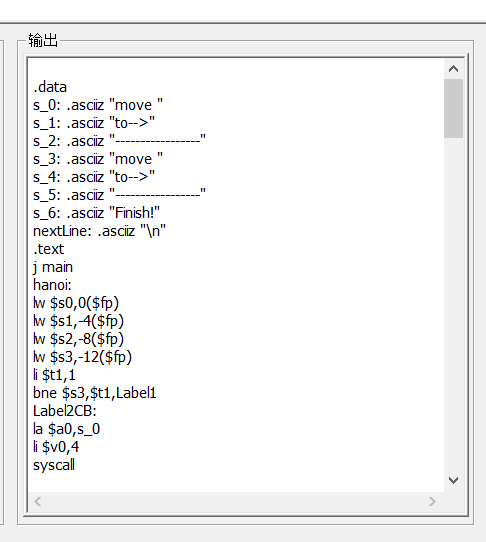
**图6 测试用例2**



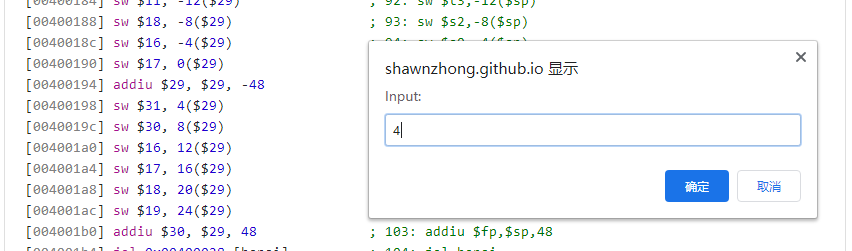
**图7 词法分析2**

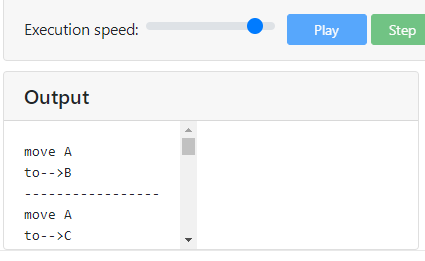


**图8 中间代码2**



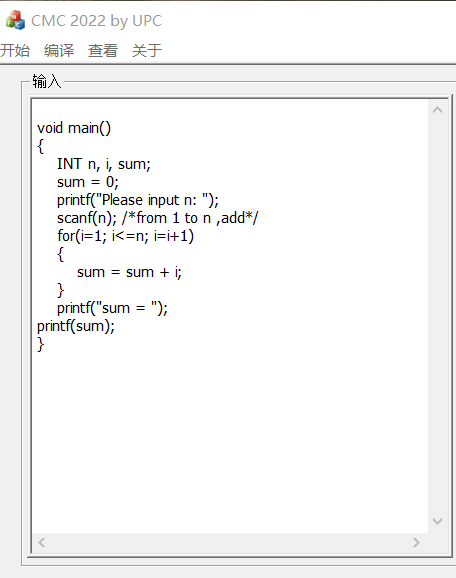
**图9 汇编代码2**



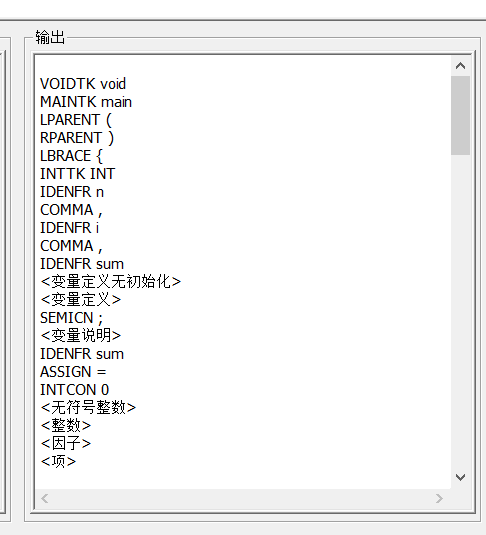


**图10 运行结果2**

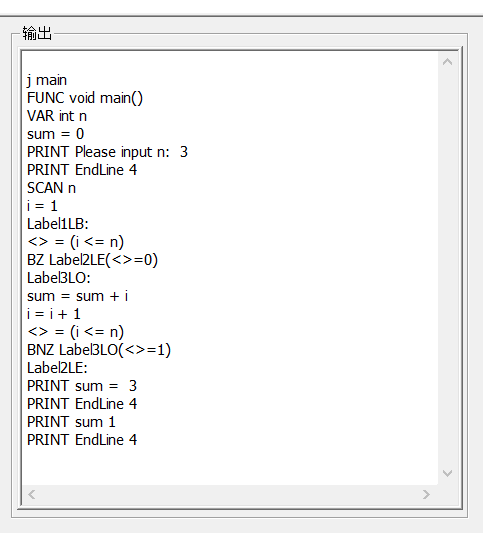
## 6.3 累加测试程序



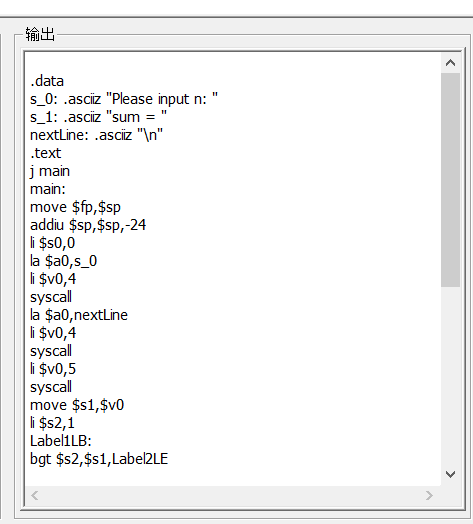
**图11 测试用例3**



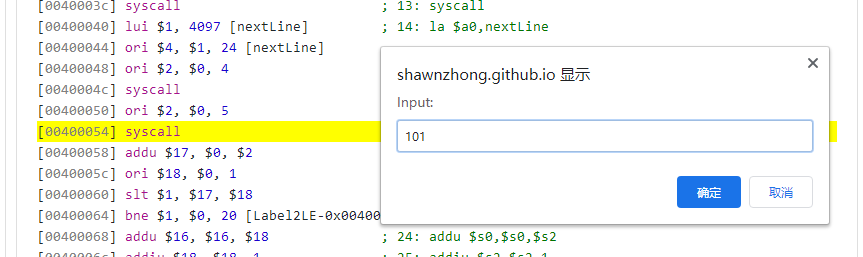
**图12 词法分析3**

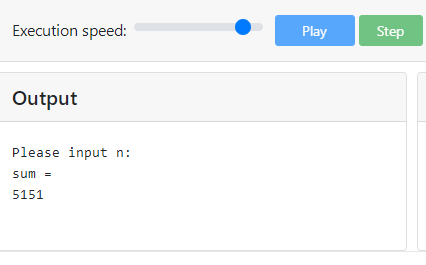


**图13 中间代码3**



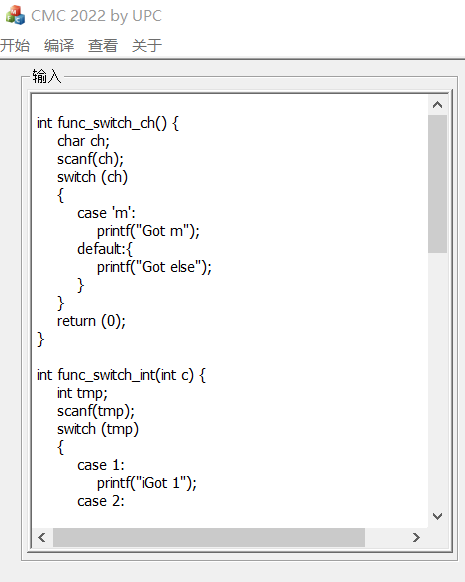
**图14 汇编代码3**



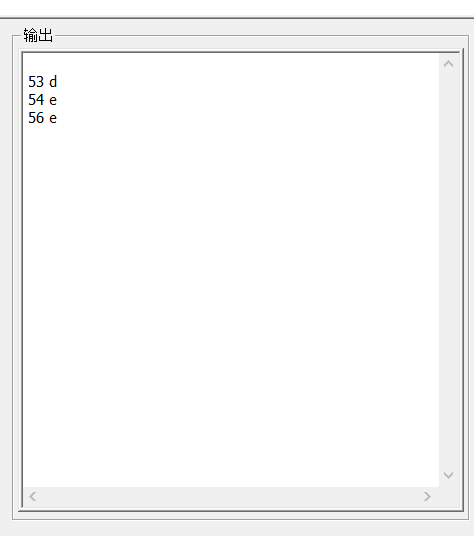


**图15 运行结果3**

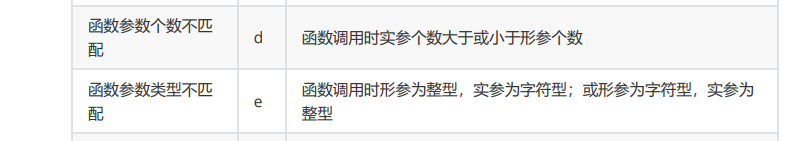
## 6.4 错误处理测试程序



**图16 测试用例4**



**图 17 输出结果**



**图18 识别到参数不匹配并报告相关错误**

# 7 心得体会

课程设计是培养学生综合运用所学知识，发现、提出、分析和解决实际问题，锻炼实践能力的重要环节，是对学生实际工作能力的具体训练和考察过程。随着科学技术发展的日新月异，计算机科学与技术已经成为当今空前活跃的领域之一，其中编译器的开发在计算机领域占据举足轻重的位置。因此作为二十一世纪的计算机专业大学生来说掌握编译器的开发技术是十分重要的。经过为期八周的编译原理课程设计，我小组成员在李村合老师的指导下，顺利完成本次课程设计要求，通过该课程设计,收获颇多：

1、化理论为实践：

经过上学期的《编译原理》课程的学习，我小组成员在课程中收获了诸多有关编译原理的知识，但是“纸上得来终觉浅”，在这学期的《编译原理课程设计》中，将理论化为实践，投入了编译器的开发学习。在课程设计过程中，不仅可以巩固了以前所学过的知识，而且学到了很多在书本上所没有学到过的知识。

通过该课程设计，掌握了什么是编译程序，编译程序工作的基本过程及其各阶段的基本任务，熟悉了编译程序总流程框图，了解了编译程序的生成过程、构造工具及其相关的技术。通过把算法的内容、算法的执行顺序在计算机上实现,理解了算法在计算机中是怎样执行的，对其在实践中的应用有深刻的理解。

同时通过本次课程设计使我们懂得了理论与实际相结合是很重要的，只有理论知识是远远不够的，只有把所学的理论知识与实践相结合起来，从理论中得出结论，才能把原来以为很深奥的书本知识变的更为简单，对实验原理有更深的理解，从而提高自己的实际动手能力和独立思考的能力。

2、化个人为团队：

在上学期的《编译原理》课程的学习过程中，我们各自为营，独自“作战”，独立学习。而在本学期的《编译原理课程设计》中，6个人凝成一股绳，分工明确、并肩“作战”。众人拾柴火焰高，一个人的力量可能是单薄的，但是团队成员之间优势互补、取长补短、各司其职，在增添了同学间凝聚力的同时，也让课程设计的工作变得轻松。

同时，在这次课程设计的团队合作中我们体会到，团队合作需要各个成员之间主动交流，积极配合，各个成员发挥自己的长处所在，完成自己的任务所属才是一个合格的团体，才会使得任务的开展事半功倍。

3、化困难为动力：

在本学期的《编译原理课程设计》中，每个人在自己负责的任务中都遇到过或多或少、或大或小的问题。产生这些问题的原因可能是理论课程知识的不足，亦或是拓展知识的不够。团队成员通过查阅资料、咨询老师、交流讨论等方式解决了开发过程中的问题，并在解决过程中巩固了学过的知识，也学习到了新的知识。解决问题的过程激发了学习的动力，增强了多学科之间的融会贯通，问题也都迎刃而解。

最终，本次课程设计顺利完成，我小组全体成员在此非常感谢李村合老师的辛勤指导与教诲，同时也感谢所有给与我们帮助的同学。最后在团队的努力下开发成功，有很强的成就感，相信在不久的将来我们会有自己的一片天空。

**教 师 评 语 专 栏**

|  |  |  |  |
| --- | --- | --- | --- |
| **任**  **课**  **教**  **师**  **评**  **语** |  | | |
| **成**  **绩** |  | **任课教师：**  **年 月 日** |