《编译技术》课程设计

文 档

学号：\_\_\_\_\_\_16231246\_\_\_\_\_\_

姓名：\_\_\_\_\_\_\_秦子柠\_\_\_\_\_\_\_

2018年11月15日

## 一．需求说明

### 1．文法说明

1. 获取的文法

＜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｝"

＜有返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'|<标识符> //第一种选择为有参数的情况，第二种选择为无参数的情况

＜无返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'|<标识符> //第一种选择为有参数的情况，第二种选择为无参数的情况

＜值参数表＞ ::= ＜表达式＞{,＜表达式＞}  
 ＜返回语句＞ ::= return['('＜表达式＞')']

＜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＞}

＜常量＞ ::= ＜整数＞|＜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＞｝

＜变量说明＞ ::= ＜变量定义＞;{＜变量定义＞;}  
 ＜变量定义＞ ::= ＜类型标识符＞(＜标识符＞|＜标识符＞'['＜无符号整数＞']'){,(＜标识符＞|＜标识符＞'['＜无符号整数＞']' )}

//＜无符号整数＞表示数组元素个数，值需大于0

＜类型标识符＞ ::= int | char

＜声明头部＞ ::= int＜标识符＞ | char＜标识符＞

＜参数表＞ ::= ＜类型标识符＞＜标识符＞{,＜类型标识符＞＜标识符＞}

＜有返回值函数定义＞ ::= ＜声明头部＞'('＜参数表＞')' '{'＜复合语句＞'}'|＜声明头部＞'{'＜复合语句＞'}'

//第一种选择为有参数的情况，第二种选择为无参数的情况

＜无返回值函数定义＞ ::= void＜标识符＞('＜参数表＞')''{'＜复合语句＞'}'| void＜标识符＞{'＜复合语句＞'}'

//第一种选择为有参数的情况，第二种选择为无参数的情况

＜复合语句＞ ::= [＜常量说明＞][＜变量说明＞]＜语句列＞  
 ＜主函数＞ ::= void main'('')''{'＜复合语句＞'}'

＜表达式＞ ::= [+|-]＜项＞{＜加法运算符＞＜项＞}//[+|-]只作用于第一个<项>  
 ＜项＞ ::= ＜因子＞{＜乘法运算符＞＜因子＞}  
 ＜因子＞ ::= ＜标识符＞|＜标识符＞'['＜表达式＞']'|'('＜表达式＞')'|＜整数＞|＜字符＞|＜有返回值函数调用语句＞  
 ＜语句＞ ::= ＜条件语句＞|＜循环语句＞|'{'＜语句列＞'}'|＜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＞;|＜空＞;|＜返回语句＞;  
 ＜语句列＞ ::= ｛＜语句＞｝

＜赋值语句＞ ::=  ＜标识符＞＝＜表达式＞|＜标识符＞'['＜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＞|＜表达式＞ //表达式为0条件为假，否则为真

＜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＞

＜读语句＞ ::= scanf '('＜标识符＞{,＜标识符＞}')'  
 ＜写语句＞ ::= printf '('＜字符串＞,＜表达式＞')'| printf '('＜字符串＞')'| printf

1. 文法改写
2. 原文法中<常量>这一成分没有在任何非终结符右部出现，故删去；
3. 原文法中<步长>这一成分仅在<循环语句>的右部出现了一次，且步长只能推出<无符号整数>这一成分，故将循环语句中的<步长>替换为无符号整数，并删去<步长>这一条规则

改写后的文法如下：

＜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

＜无符号整数＞ ::= ＜数字＞｛＜数字＞｝

＜整数＞ ::= ［＋｜－］＜无符号整数＞

＜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｝"

＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝

＜程序＞::={const＜常量定义＞;}{(int|char)＜标识符＞['['＜无符号整数＞']']{<连续变量定义>};}{(int|char|void)＜标识符＞('('<有参数函数定义片段> | '{'<无参数函数定义片段>)}＜主函数＞

<无参数函数定义片段> ::= ＜复合语句＞'}'

<有参数函数定义片段> ::= ＜参数表＞')' '{'＜复合语句＞'}'

<连续变量定义> ::= ,(＜标识符＞|＜标识符＞'['＜无符号整数＞']')

＜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＞}

＜变量定义＞ ::= (int | char)(＜标识符＞|＜标识符＞'['＜无符号整数＞']'){<连续变量定义>}

//＜无符号整数＞表示数组元素的个数，其值需大于0

＜复合语句＞::= {const＜常量定义＞;}{＜变量定义＞;}＜语句列＞

＜参数表＞ ::= (int | char)＜标识符＞{,(int | char)＜标识符＞}

＜主函数＞::= void main'('')''{'＜复合语句＞'}'

＜表达式＞ ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞} //[+|-]只作用于第一个<项>

＜项＞ ::= ＜因子＞{＜乘法运算符＞＜因子＞}

＜因子＞::= ＜标识符＞｜＜标识符＞'['＜表达式＞']'|'('＜表达式＞')'｜＜整数＞|＜字符＞｜＜有返回值函数调用语句＞

＜语句＞ ::= ＜条件语句＞｜＜循环语句＞| '{'＜语句列＞'}'| ＜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＞;｜＜空＞;|＜返回语句＞;

＜赋值语句＞::= (＜标识符＞|＜标识符＞'['＜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＞｜＜表达式＞

//表达式为0条件为假，否则为真

＜循环语句＞::= (while'('＜条件＞')'|for'('＜标识符＞＝＜表达式＞;＜条件＞;＜标识符＞＝＜标识符＞(+|-)＜无符号整数＞')')＜语句＞

＜有返回值函数调用语句＞::=＜标识符＞'('＜值参数表＞')'|<标识符> //第一种选择为有参数的情况，第二种选择为无参数的情况

＜无返回值函数调用语句＞::=＜标识符＞'('＜值参数表＞')'|<标识符> //第一种选择为有参数的情况，第二种选择为无参数的情况

＜值参数表＞ ::= ＜表达式＞{,＜表达式＞}

＜语句列＞ ::= ｛＜语句＞｝ //0-inf个语句

＜读语句＞ ::= scanf '('＜标识符＞{,＜标识符＞}')'

＜写语句＞::= printf'('(＜字符串＞,＜表达式＞ | ＜字符串＞ | ＜表达式＞) ')'

＜返回语句＞ ::= return['('＜表达式＞')']

虽然<程序>成分中仍有头符号交集不为空，但是这样改写后的文法在编程实现上的难度相对之前已经有所下降，仅需使用一个变量指示当前是否在函数定义部分即可正确判断变量定义部分是否结束。除此之外，其他的语法成分也有不同程度的简化，与递归下降分析法适应性较好。

【说明获取的文法、对文法的改写和扩充】

### 2．目标代码说明

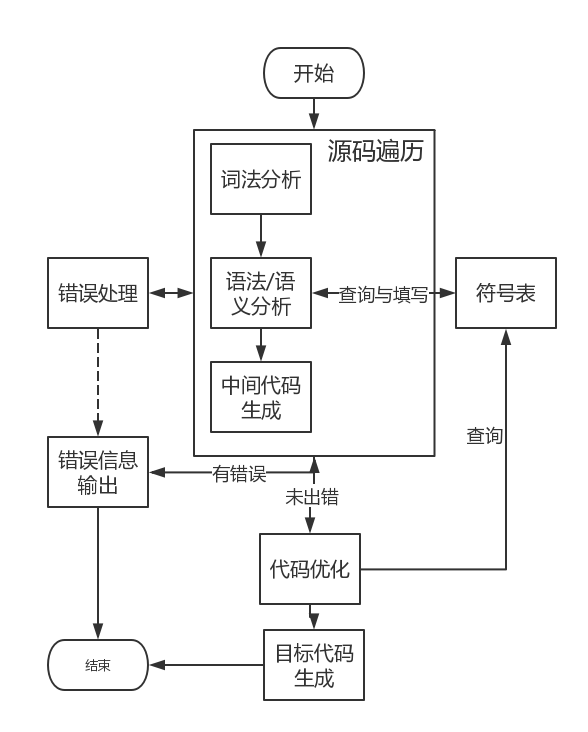
【说明要生成的目标代码指令及含义】

### 3. 优化方案\*

【说明需要完成的优化方案及其要求】

## 二．详细设计

### 1．程序结构



### 2．函数结构、功能以及相互调用关系

为了防止不合法的调用，在头文件中定义命名空间对全局变量以及函数进行作用域的控制。

词法分析：

外部不可见的函数和变量：

宏函数：BACK\_1\_CHAR，isALPHA(c)，isWHITE(c)，isSGL\_CHAR(c)，isSGL\_QTE(c)，isDBL\_QTE(c)，分别为退读一个字符，判断字符是否是文法中的<字母>，判断字符是否是空格/制表/换行，判断字符是否是定义字符常量时的字符集中的一个字符，判断字符是否是单引号，判断字符是否是双引号。

代码列索引std::size\_t linePointer，指示处理的当前行的代码的代码位置，即下次取词从何处开始。

代码行std::string codeLine，存储当前的一行代码。

命名空间为Lex，可供调用的函数/全局变量/常量有：

int getsym()

该函数获取一个单词，以字符串的形式存到并将其类别作为返回值，返回值为以下的表达式常量：

extern constexpr auto UNKNOWN = 0; // 5StarsBB

extern constexpr auto IDEN = 1; // ISLAND

extern constexpr auto UNSGN\_INT = 2; // 19260817

extern constexpr auto SGL\_CHARA = 3; // 'a'

extern constexpr auto STRING = 4; // "HuaLaiShi"

extern constexpr auto RSVD\_WD = 5; // if

extern constexpr auto RSVD\_SYM = 6; // ==

保留字和非字母的关键符号到助记符的映射表，用于词法分析阶段的调试输出。其中的内容已在注释中阐明。

extern const std::map<std::string, std::string> reservedWords;/\* = {

{"const", "CONST\_SYM"},

{"int", "INT\_SYM"},

{"char", "CHAR\_SYM"},

{"void", "VOID\_SYM"},

{"main", "MAIN\_SYM"},

{"if", "IF\_SYM"},

{"else", "ELSE\_SYM"},

{"for", "FOR\_SYM" },

{"while", "WHILE\_SYM"},

{"scanf", "SCANF\_SYM"},

{"printf", "PRINTF\_SYM"},

{"return", "RET\_SYM"}

};\*/

extern const std::map<std::string, std::string> non\_alpha\_sym; /\*=

{

{"+", "ADD"}, {"-", "SUB"},

{"\*", "MUL"}, {"/", "DIV"},

{"(", "L\_SMALL"}, {")", "R\_SMALL"},

{"[", "L\_MID"}, {"]", "R\_MID"},

{"{", "L\_BIG"}, {"}", "R\_BIG"},

{"=", "ASSIGN"},

{"!=", "NOT\_EQL"}, {"==", "EQUAL"},

{">", "GREATER"}, {">=", "GRT\_EQL"},

{"<", "LESS"}, {"<=", "LES\_EQL"},

{"\"", "DBL\_QUOTE"},{"'", "SGL\_QUOTE"},

{",", "COMMA"}, {";", "SEMICOLON"}

};\*/

代码文件输入流：

extern std::ifstream code\_file;

行号记录，用于错误处理时的信息记录：

extern int LineCounter;

语法分析/语义分析：命名空间Lex在其中可见。

在进行新的语法成分分析前，确保Lex::curElmt为新的word：在错误处理时进行word的跳过，若没有出错则在返回前调用Lex中的getsym()将新的word读到curElmt中。

【描述各类/方法或函数的功能，以及关键算法】

### 3．符号表管理方案

文法显示，常量、变量的定义仅出现在整个程序开头和main函数体的开头，因此只需构造全局符号表和函数符号表。

使用C++内置的map容器实现这两个表，其底层实现有hash过程，因此无需另外手工实现hash表。

全局符号表：用一个string到struct varInfo的map实现，键为字符串存储的标识符名称，struct varInfo将在下文说明，它存储的是该标识符的信息。

函数符号表：用一个string到map<string, struct varInfo>的复合map实现，键为函数名，值的键为该函数内的局部符号名称，值的值varInfo存储的该局部符号的信息。

Struct varInfo：设计暂未完善

查表时先在函数符号表中查询，没有再查全局符号表。

另外对于字符串常量，用一个set记录所有的字符串。另外，在

【说明符号表的数据结构、管理算法】

### 4．存储分配方案

静态分配：全局变量通过$gp进行索引，全局常量全部优化成数值。字符串常量用.asciiz字段在.text前放在.data区域。

每个函数调用的运行栈从高到低依次为：（其实是按照$1-$31的顺序，但是没有将$t1-$t9压入栈中）

|  |
| --- |
| 参数区 |
| 局部变量区 |
| 寄存器栈（保存调用者的寄存器的情况） |
| 上一次的帧指针，即$fp寄存器 |
| 上一次的$ra寄存器 |

【说明运行时的存储组织及管理方案，运行栈结构】

### 5. 四元式设计\*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| op | op含义 | 操作数1 | 操作数2 | 结果操作数 |
| REALPARAOP | push | 和结果操作数一样 | 0 | 可能是标识符或者是数字 |
| READ\_ARRAY\_OP | =[]  a = b[1]; | 数组标识符 | 下标操作数 | 要赋值的操作数 |
| CALL\_OP | call  表示调用这个函数 |  |  | 函数标识符 |
| J\_OP | goto  表示无条件跳转 |  |  | label |
| FUNCBEGINOP | init  表示这个函数的开始 |  |  | 函数标识符 |
| RETOP | ret  表示这个函数的结束 |  |  | 函数标识符 |
| MULOP | \* | 标识符或数字 | 标识符或数字 | 标识符 |
| DIVOP | / | 标识符或数字 | 标识符或数字 | 标识符 |
| ADDOP | + | 标识符或数字 | 标识符或数字 | 标识符 |
| SUBOP | - | 标识符或数字 | 标识符或数字 | 标识符 |
| EQUALOP, NEQOP, MOREOP, MOREOREQUOP, LESSOP, LESSOREQUOP, | ==  !=  >  >=  <  <=  分别表示相应的条件跳转语句 | 标识符或数字 | 标识符或数字 | label |
| SCANFOP | scanf  输入 | 结果操作数 | int/char | 标识符 |
| PRINTFOP | printf  （输出表达式） | 结果操作数 | int/char | 标识符 |
| PRINTFOP | printf  （输出字符串） | 结果操作数 | 0 | 字符串在字符串表中的位置 |
| LOAD\_ARR\_OP | []=  a[1] = b; | 给数组值的操作数 | 下标操作数 | 数组标识符 |
| LABOP | label:  表示标签 | 0 | 0 | label |
| EXITOP | 表示程序结束 |  |  |  |

【对采用的四元式进行详细说明】

### 6. 目标代码生成方案\*

【说明代码生成有关的数据结构、关键算法】

### 7. 优化方案\*

基本块内部的公共子表达式删除（DAG图）；

全局寄存器分配（着色算法）。

窥孔优化

最优指令选择：

1. 乘除1不操作，加减0不操作，乘除2的n次幂时（n为正整数），用符号左移/右移代替mul/div与mfhi/mflo的组合

【说明代码优化有关的数据结构、关键算法】

### 8. 出错处理

词法分析阶段：  
 字符常量缺少右双引号

字符常量中的字符太多

字符常量中的字符有非法字符

字符串常量缺少右双引号

单独出现!

未知符号

语法分析阶段：

不是 ;

不是一个标识符

不是 =

不是 int或char保留字

不是 ]

不是 )

不是 一个<因子>的开始单词

不是 {

不是 }

不是 main

不是 (

不是一个非0无符号类型

不是 一个<语句>的开始单词

不是 else

不是 常量

非法步长

不是 变量定义的开始符号

语义分析阶段：

标识符重定义

标识符未定义

参数类型不匹配

参数数量不匹配

不是 int类型的表达式

不是 数组类型的标识符

不是 函数名

不是 一个无参函数

不是 一个被正确使用的标识符

不是 一个简单变量

有返回值的函数没有一句return

void类型的函数return了一个实际值

return的类型不匹配

不允许int到char的转换

不是 一个有返回值的函数

【说明出错处理方案、错误信息及含义】

## 三．操作说明

### 1．运行环境

【说明搭建运行环境的步骤】

### 2．操作步骤

【详细说明操作步骤】

## 四．测试报告

### 1．测试程序及测试结果

【给出提供的测试程序以及每个程序的测试结果，至少5个正确程序，5个错误程序，无需截屏】

### 2．测试结果分析

【说明上述测试程序对语法成分的覆盖情况】

## 五．总结感想

【说明在完成课程设计中的收获、认识和感想】