《编译技术》

课程设计文档

学号：\_\_\_\_\_13071079\_\_\_\_\_

姓名：\_\_\_\_\_\_\_\_卢嵩\_\_\_\_\_\_\_\_

2016年 11 月 10 日

# 需求说明

## 文法说明

### 获取的完整文法

＜加法运算符＞ ::= +｜-

＜乘法运算符＞ ::= \*｜/

＜关系运算符＞ ::= <｜<=｜>｜>=｜!=｜==

＜字母＞ ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

＜数字＞ ::= ０｜＜非零数字＞

＜非零数字＞ ::= 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｝"

＜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＞}

＜无符号整数＞ ::= ＜非零数字＞｛＜数字＞｝

＜整数＞ ::= ［＋｜－］＜无符号整数＞｜０

＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝

＜声明头部＞ ::= int＜标识符＞ |char＜标识符＞

＜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＞‘]’ )}

＜常量＞ ::= ＜整数＞|＜字符＞

＜类型标识符＞ ::= int | char

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

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

＜复合语句＞ ::= ［＜常量说明＞］［＜变量说明＞］＜语句列＞

＜参数＞ ::= ＜参数表＞

＜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＞‘]’|‘(’＜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＞;|＜情况语句＞｜＜返回语句＞;

＜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＞

＜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＞

＜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＞|＜空＞

＜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＞{,＜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＞‘)’]

附加说明：

（1）char类型的表达式，用字符的ASCII码对应的整数参加运算，在写语句中输出字符

（2）标识符区分大小写字母

（3）写语句中的字符串原样输出

（4）情况语句中，switch后面的表达式和case后面的常量只允许出现int和char类型；每个情况子语句执行完毕后，不继续执行后面的情况子语句

（5）数组的下标从0开始

### 文法改写

为简化设计编译器，本人对原始文法进行了部分改写，主要改写部分采用红色加粗表示。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｜＜非零数字＞

＜非零数字＞ ::= 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｝"

＜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＞}

＜无符号整数＞ ::= ＜非零数字＞｛＜数字＞｝

＜整数＞ ::= ［＋｜－］＜无符号整数＞｜０

＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝

＜声明头部＞ ::= int＜标识符＞ |char＜标识符＞

＜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＞‘]’ )}

＜常量＞ ::= ＜整数＞|＜字符＞

＜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＞‘]’|‘(’＜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＞;|＜情况语句＞｜＜返回语句＞;

＜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＞

＜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＞

＜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＞|＜空＞

＜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＞{,＜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＞‘)’]

改动总结：删除<参数>项，使用＜参数表＞代替＜参数＞

### 文法扩充说明：

为了输出美观，printf结束后默认换行。

## 目标代码说明

【说明要生成的目标代码指令及含义】

要生成的目标代码为32位MIPS汇编代码，具体包含的指令如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 指令 | 助记符 | 示例 | 含义 |
| 加 | add | add $1,$2,$3 | $1 = $2 + $3 |
| 减 | sub | sub $1,$2,$3 | $1 = $2 - $3 |
| 乘 | mult | mult $1, $2 | Hi,Lo = $1 \* $2 |
| 除 | div | div $s, $t | $LO = $s / $t;  $HI = $s % $t; |
| 立即数加 | addi | addi $t, $s, 1406 | $t = $s + imm |
| 取字 | lw | lw $t, 4($s) | $t = MEM[$s + 4] |
| 存字 | sw | sw $t, offset($s) | MEM[$s + offset] = $t; |
| 条件分支 | beq | beq $s, $t, offset | if $s == $t advance\_pc (offset << 2)) ; else advance\_pc (4); |
| 条件分支 | bne | bne $s, $t, offset | if $s != $t advance\_pc (offset << 2)) ; else advance\_pc (4); |
| 无条件跳转 | j | j target | PC = nPC; nPC = (PC & 0xf0000000) | (target << 2); |
| 跳转并返回 | jr | Ja $ra | Goto $ra |
| 系统调用 | syscall | syscall | 执行相关系统函数 |

## 优化方案\*

常数合并，DAG消除基本块内部的公共子表达式，寄存器分配优化，循环优化等

# 详细设计

## 程序结构

程序大体上分为8个文件，其中有7个cpp文件，一个头文件。具体结构如下图，执行的顺序由箭头所示。其中global.h中包含了所有宏定义以及所有执行所需要用到的函数定义。程序接受到输入后首先执行词法分析程序，词法分析完成以后执行语法分析，确定输入的语言成分，完成建立符号表等相关操作。在进行词法和语法分析的过程中可能会调用error.h来进行错误信息的处理。在完成语法分析之后，程序将会生成中间代码，中间代码以四元式的形式表示，具体的四元式形式见后文。在完成四元式生成之后，我们的程序将会把四元式转化为mips汇编程序并做输出，完成编译工作。在进行中间代码生成和汇编代码生成的时候需要同时兼顾代码的优化，尽量提高生成代码的质量。



## 类/方法/函数功能

各类的功能如上图所示，再次不做赘述。以下列举全部函数及其功能：

### 词法分析：

|  |  |
| --- | --- |
| 函数 | 功能 |
| void getpath() | 用来获取输入文件的路径 |
| void clearToken() | 用来清空token |
| void getnextline() | 获取下一行源代码的输入 |
| void *getch*() | 获得下一个输入字符，当读到文件末尾的时候返回空字符 |
| void catToken() | 将当前字符与token拼接 |
| void skip() | 跳过无意义的字符，本文法中针对空格，换行符和制表符 |
| bool isLetter() | 判断当前ch是否是字母类型，注意根据文法定义下划线'\_'也算作字母 |
| bool isDigit() | 判断ch是否是数字 |
| void retract() | 读字符的指针退格 |
| int isreserve() | 分辨当前标识符是否是保留字，是的话返回对应的sym，不是返回0说明是标识符 |
| bool isStringCon() | 检查当前ch是不是组成字符串的合法元素，文法定义为“十进制编码为32,33,35-126的ASCII字符” |
| int getsym() | 词法分析主要部分，判别当前单词的类型 |
| void output() | 将分析结果输出，为下一步语法分析做准备 |
| int lexer() | 词法分析执行的主函数 |

### 语法分析

## 调用依赖关系

【说明各类之间的关系，方法/函数之间的调用关系】

## 符号表管理方案

【说明符号表的数据结构、管理算法】

符号表拟采用栈式存储结构

## 存储分配方案

【说明运行时的存储组织及管理方案，运行栈结构】

## 解释执行程序\*

【说明解释执行程序的数据结构，关键算法，及解释执行过程】

## 四元式设计\*

【对采用的四元式进行详细说明】

## 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】

## 优化方案\*

【说明代码优化有关的数据结构、关键算法】

## 出错处理

【说明出错处理方案、错误信息及含义】

# 操作说明

## 运行环境

【说明搭建运行环境的步骤】

## 操作步骤

【详细说明操作步骤】

# 测试报告

## 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】

## 测试结果分析

【说明上述测试程序对语法成分的覆盖情况】

# 总结感想

【说明在完成课程设计中的收获、认识和感想】

注：【】内的文字为文档模板说明，完成的作业中需去掉。

标\*的章节需根据题目的难度进行取舍。