《编译技术》课程设计文 档

学号：\_\_\_\_\_\_\_\_\_16061182\_\_\_\_\_\_\_\_\_\_\_

姓名：\_\_\_\_\_\_\_\_\_苗萌\_\_\_\_\_\_\_\_\_\_\_

2018 年 11 月 17 日

**一．需求说明**

**1．文法说明**

文法符合要求。

有些比较简单的非终结符直接在词法分析部分作为单个符号处理，不再作为语法分析的分析函数，这些非终结符已经用红色高亮显示。

此外，为了便于语法分析的分析，新增了一个非终结符＜外部变量说明＞，其文法规则和＜变量说明＞相同，已经用蓝色高亮显示。

＜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｝"  
1、＜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＞  
2、＜常量说明＞ ::=  const＜常量定义＞;{ const＜常量定义＞;}  
3、＜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＞  ::= ＜数字＞｛＜数字＞｝  
4、＜整数＞        ::= ［＋｜－］＜无符号整数＞  
＜标识符＞    ::=  ＜字母＞｛＜字母＞｜＜数字＞｝  
＜声明头部＞   ::=  int＜标识符＞ |char＜标识符＞  
5、＜外部变量说明＞  ::= ＜变量定义＞;{＜变量定义＞;}

6、＜变量说明＞  ::= ＜变量定义＞;{＜变量定义＞;}  
7、＜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＞表示数组元素的个数，其值需大于0  
＜类型标识符＞      ::=  int | char  
8、＜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  
9、＜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  
10、＜复合语句＞   ::=  ［＜常量说明＞］［＜变量说明＞］＜语句列＞  
11、＜参数表＞    ::=  ＜类型标识符＞＜标识符＞{,＜类型标识符＞＜标识符＞}  
12、＜主函数＞    ::= void main‘(’‘)’‘{’＜复合语句＞‘}’  
13、＜表达式＞    ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞}  //[+|-]只作用于第一个<项>  
14、＜项＞     ::= ＜因子＞{＜乘法运算符＞＜因子＞}  
15、＜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＞  
16、＜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＞;  
17、＜赋值语句＞   ::=  ＜标识符＞＝＜表达式＞|＜标识符＞‘[’＜表达式＞‘]’=＜表达式＞  
18、＜条件语句＞::= if ‘(’＜条件＞‘)’＜语句＞[else＜语句＞]  
19、＜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  
20、＜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＞  
21、＜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  
22、＜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  
23、＜值参数表＞   ::= ＜表达式＞{,＜表达式＞}  
24、＜语句列＞   ::= ｛＜语句＞｝  
25、＜读语句＞    ::=  scanf ‘(’＜标识符＞{,＜标识符＞}‘)’  
26、＜写语句＞    ::= printf ‘(’ ＜字符串＞,＜表达式＞ ‘)’| printf ‘(’＜字符串＞ ‘)’| printf ‘(’＜表达式＞‘)’  
27、＜返回语句＞   ::=  return[‘(’＜表达式＞‘)’]    
  
附加说明：  
（1）char类型的变量或常量，用字符的ASCII码对应的整数参加运算  
（2）标识符不区分大小写字母  
（3）写语句中，字符串原样输出，单个字符类型的变量或常量输出字符，其他表达式按整型输出   
（4）数组的下标从0开始  
（5）for语句先执行一次循环体中的语句再进行循环变量是否越界的测试

（6）允许++4、--4、+-4、-+4的表示出现，前两种结果为正整数4，后两种结果为负整数4

**2．目标代码说明**

目标代码为MIPS指令，种类如下所示（[$s2]表示以$s2为地址保存的字）

* syscall，系统调用，不同的$v0代表不同的功能

2、li $s1,2，记录立即数，$s1 = 2

3、la $s1,list1，取地址，$s1 = list1的首地址

4、add $s1,$s2,$s3，加法，$s1 = $s2 + $s3

5、addi $s1,$s2,2，立即数加法，$s1= $s2 + 2

6、sub $s1,$s2,$s3，减法，$s1 = $s2 - $s3

7、subi $s1,$s2,2，立即数减法，$s1 = $s2 - 2

8、mult $s1,$s2，乘法，$hi,$lo = $s1 \* $s2

9、div $s1,$s2，除法，$lo = $s1 / $s2,$hi = $s1 % $s2

10、mfhi $s1，取$hi寄存器的值，$s1 = $hi

11、mflo $s1，取$lo寄存器的值，$s1 = $lo

12、move $s1,$s2，赋值，$s1 = s2

13、lw $s1,0($s2)，取字，$s1 = [$s2]

14、sw $s1,0($s2)，存字，[$s2] = $s1

15、beq $s1,$s2,label，相等则跳转，$s1 == $s2 => goto label

16、bne $s1,$s2,label，不等则跳转，$s1 != $s2 => goto label

17、blt $s1,$s2,label，小于则跳转，$s1 < $s2 => goto label

18、bgt $s1,$s2,label，大于则跳转，$s1 > $s2 => goto label

19、ble $s1,$s2,label，小于等于则跳转，$s1 <= $s2 => goto label

20、bge $s1,$s2,label，大于等于则跳转，$s1 >= $s2 => goto label

21、j label，无条件跳转，goto label

22、jr $ra，跳转到指定位置，pc = $ra

23、jal label，无条件跳转并链接，$ra = pc + 4,goto label

**3. 优化方案\***

1、常数合并：表达式计算时，若已知部分操作数的值，可以先对这些值进行预处理，之后再与变量一起计算

2、常量替换：在编译的时候把常量替换为数字，之后就不再对其进行取值

3、消除公共子表达式：划分基本块，删除公共子表达式

4、全局寄存器分配：引用计数算法和着色图算法

5、数据流分析：活跃变量分析，定义和使用链

6、窥孔优化：在较短的指令序列中，删除冗余或者低效率的指令，保证最终指令序列简洁而高效

**二．详细设计**

**1．程序结构**

程序分为六个部分：词法分析、语法和语义分析、中间代码生成、目标代码生成、错误处理、优化。

语法和语义分析部分调用词法分析部分，并调用中间代码生成部分生成中间代码，若在此过程中遇到错误，则调用错误处理部分。之后调用目标代码生成部分来生成目标代码，最后调用优化部分来优化目标代码。

**2．类/方法/函数功能**

词法分析：

1、void getch()，读取单个字符

2、void getsym()，读取一个符号

语法分析和语义分析：

void program(),程序子程序

* void deccon()，常量说明子程序
* void defcon()，常量定义子程序
* void integer()，整数子程序
* void decvar\_extern()，外部变量说明子程序
* void decvar()，变量说明子程序
* void defvar()，变量定义子程序
* void deffunct()，有返回值函数定义子程序
* void deffuncf()，无返回值函数定义子程序
* void multistate()，复合语句子程序
* void paralist()，参数表子程序
* void mainfunc()，主函数子程序
* void expr()，表达式子程序
* void item()，项子程序
* void factor()，因子子程序
* void state()，语句子程序
* void fuzhistate()，赋值语句子程序
* void tiaojianstate()，条件语句子程序
* void tiaojian()，条件子程序
* void xunhuanstate()，循环语句子程序
* void callfunctstate()，有返回值函数调用语句子程序
* void callfuncfstate()，无返回值函数调用语句子程序
* void valueparalist()，值参数表子程序
* void states()，语句列子程序
* void readstate()，读语句子程序
* void writestate()，写语句子程序
* void returnstate()，返回语句子程序

中间代码生成：

* void getmid()，得到中间代码

优化部分：

* void conbcon()，常数合并
* void replcon()，常数替换
* void delexpr()，删除公共子表达式
* void allocreg()，分配全局寄存器
* void datastream()，数据流分析
* void delcode()，删除无用代码

目标代码生成：

* void printcode()，根据中间代码得到mips指令

错误处理：

1、void error()，错误处理

**3．调用依赖关系**

语法分析、语义分析函数会调用getsym()读取符号，getsym()会调用getch()读取字符；getsym()、语法和词法分析函数报错时会调用error()。除此之外，其他函数调用关系如下：

program()调用deccon()，decvar()，deffunct()，deffuncf()，mainfunc()

* deccon()调用defcon()
* defcon()无
* integer()无
* decvar\_extern()调用defvar()
* decvar()调用defvar()
* defvar()无
* deffunct()调用dechead()，paralist()，multistate()
* deffuncf()调用paralist()，multistate()
* multistate()调用deccon()，decvar()，states()
* paralist()，无
* mainfunc()，调用multistate()
* expr()，调用item()
* item()，调用factor()
* factor()，调用expr()，integer()，callfunctstate()
* state()，调用tiaojianstate()，xunhuanstate()，states()，callfunctstate()，callfuncfstate()，fuzhistate()，readstate()，writestate()，returnstate()
* fuzhistate()，调用expr()
* tiaojianstate()，调用tiaojian()，state()
* tiaojian()，调用expr()
* xunhuanstate()，调用tiaojian()，state()，expr()
* callfunctstate()，调用valueparalist()
* callfuncfstate()，调用valueparalist()
* valueparalist()，调用expr()
* states()，调用state()
* readstate()，无
* writestate()，调用expr()
* returnstate()，调用expr()

**4．符号表管理方案**

符号表的数据结构（无序符号表）：

**typedef** struct**{**

char name**[**IDENL**];**//符号id

int type**;**//符号种类,0:const,1:var,2:func,3:para

int value**;**//符号值,如果符号是函数名,则0:函数为int型,1:函数为char型,2:函数为void型

int address**;**//标识符存储地址或者相对地址

int paranum**;**//函数:表示参数个数,数组:表示数组大小

int level**;**//作用域的嵌套层次

**}**symbol**;**

**typedef** struct**{**

symbol symbols**[**MAXTAB**];**//符号表

int index**;**//符号表栈顶指针

int totalblock**;**//记录当前符号表拥有的分程序总数

int blockindex**[**MAXPRE**];**//分程序的索引

**}**tab**;**

符号表管理算法：

用tab来记录符号表，存储不同层次的符号表，使用findsym()函数来判断当前变量是否在符号表中声明过，使用pushsym()来将符号插入符号表，另有一系列函数用来给symbol的域赋值。

**5．存储分配方案**

运行栈按照课本示例设计，如下图所示：

|  |
| --- |
| 局部参数区 |
| 参数区  形参数据区（显式参数区） |
| prev abp（调用模块基地址） |
| ret addr（下一条指令地址） |
| ret value（返回值） |
| display区 |
| …… |
| …… |
| 主函数变量定义区 |

采用固定寄存器分配方案，共使用三个寄存器，两个是从内存中读入的数，一个是计算结果，然后再把计算结果存入内存。

**6. 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】

**7. 四元式设计\***

四元式

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 描述 | F1 | F2 | F3 | F4 | 示例 |
| 常量定义 | const | int/char | name | value | const int a 0 |
| 变量定义 | var | int/char | name | 0 | var int a 0 |
| 数组定义 | array | int/char | name | length | array int a 5 |
| 函数定义 | func | int/char/void | name | paranum | func int func 1 |
| 函数调用 | call | int/char/void | name | paranum | call int func 1 |
| 函数定义参数 | para | int/char | name |  | para int a |
| 函数调用参数（常数） | push\_con | int/char |  | value | push\_con int 1 |
| 函数调用参数（变量） | push\_var | int/char | name |  | push\_var int a |
| 赋值（常数） | assign\_con | int/char | name | value | assign\_con char c 1 |
| 赋值（变量） | assign\_var | int/char | name | varname | assign\_var char c a |
| 数组元素赋值（常数） | assign\_arr\_con | name | index | value | assign\_arr\_con a 1 0 |
| 数组元素赋值（变量） | assign\_arr\_var | name | index | varname | assign\_arr\_var a 1 c |
| 数值运算 | +-\*/ | t1 | t2 | out | + t1 t2 out |
| 常数因子 | factor\_con | int/char | t1 | value | factor\_con int t1 1  (in1 = 1) |
| 变量因子 | factor\_var | int/char | t1 | varname | factor\_var int t1 a |
| 函数因子 | FFACTOR | INT/CHAR | in1 | $v0 | FFACTOR INT in1 $v0 |
| 数组因子 | AFACTOR | in1 | name | index | AFACTOR in1 array 0 |
| 等于jump | beq | in1 | in2 | label | beq in1 in2 mark |
| 不等jump | bne | in1 | in2 | label | bne in1 in2 mark |
| 大于jump | bgtr | in1 | in2 | label | bgtr in1 in2 mark |
| 大于等于jump | bgeq | in1 | in2 | label | bgeq in1 in2 mark |
| 小于jump | blss | in1 | in2 | label | blss in1 in2 mark |
| 小于等于jump | bleq | in1 | in2 | label | bleq in1 in2 mark |
| 无条件jump | jump | label |  |  | jump mark |

**8. 目标代码生成方案\***

目标代码（mips指令）是由中间代码（四元式）生成的。目标代码生成的主要任务包括：从中间代码到目标代码的转换过程中进行指令选择，目标代码地址空间的划分，目标体系结构上寄存器和内存单元的分配和指派，目标代码序列的生成等。

**9. 优化方案\***

其他优化方案要等到实现时再考虑是否采用，因此先只写一个消除公共子表达式：

1. 初始化一个放置 DAG图中间结点的队列。

2. 如果DAG图中还有中间节点未进入队列，则执行步骤 3，否则执行步骤 5。

3. 选取一个尚未进入队列，但其 所有父节点均已进入队列 的中间节点n，将其加入队列；或选取 没有父节点的中间节点，将其加入队列。

4. 如果n的最左子节点符合步骤3的条件，将其加入队列；并沿着当前节点的最左边，循环访问其最左子节点，最左子节点的最左子节点等，将符合步骤3条件的中间节点依次加入队列；如果出现不符合步骤 3条件的最左子节点，执行步骤 2。

5. 将中间节点队列逆序输出，便得到中间节点的计算顺序，将其整理成中间代码序列。

**10. 出错处理**

处理方式：输出错误发生的位置和错误的种类

错误种类如下：

* 文件打开失败
* 引用未定义函数名
* 引用未定义标识符
* 表达式语法错误
* 数值应该是整数
* 缺少;
* 缺少,
* 缺少一对()
* 缺少一对[]
* 缺少一对{}
* 缺少(
* 缺少)
* 缺少[
* 缺少]
* 缺少{
* 缺少}
* 数组越界
* 数组长度为0
* 函数名冲突
* 变量名冲突
* 局部变量名和宏定义冲突
* 在函数的声明头部部分没有参数名
* 在常量声明时没有给常量赋值
* 常量或变量声明时类型不为int或char
* 常量或变量声明时缺省int或char
* 非法字符
* 缺少预定义关键字
* 除数为0
* 缺少返回语句
* 函数返回值类型错误
* 赋值时等号两边类型不匹配或类型错误
* 函数调用时传入的参数错误

**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，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。