（附录）

《编译技术》课程设计文档

学号：\_\_\_\_\_\_\_\_15061062\_\_\_\_\_\_\_\_\_\_\_\_

姓名：\_\_\_\_\_\_\_\_缪佳敏\_\_\_\_\_\_\_\_\_\_\_\_

2018年 1月 13日

## 一．需求说明

### 1．文法说明

1. ＜加法运算符＞ ::= +｜-

范例：+

分析：定义了加法运算符，包括+和-，两者只能出现其中一个。

1. ＜乘法运算符＞ ::= \*｜/

范例：\*

分析：定义了乘法运算符，包括\*和/，两者只能出现其中一个。

1. ＜关系运算符＞ ::= <｜<=｜>｜>=｜!=｜==

范例：<

分析：定义了关系运算符，包括<，<=，>，>=，!=，==，只能单独出现其中一个，不允许为空。

1. ＜字母＞ ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

范例：a

分析：定义了字母，包括下划线\_，小写字母a-z，大写字母A-Z，只能单独出现其中的一个，不允许为空。

1. ＜数字＞ ::= ０｜＜非零数字＞

范例：0

分析：定义了数字，由0或非零数字组成，只能出现其中一个，不允许为空。

1. ＜非零数字＞ ::= １｜．．．｜９

范例：5

分析：定义了非零数字，由1-9组成，只能出现其中一个，不允许为空。

1. ＜字符＞ ::= '＜加法运算符＞'｜'＜乘法运算符＞'｜'＜字母＞'｜'＜数字＞'

范例：’+’

‘a’

‘5’

分析：定义了字符，由＜加法运算符＞，＜乘法运算符＞，＜字母＞，＜数字＞用单引号’’引起来组成，只能出现其中一个，不允许为空。

1. ＜字符串＞ ::= "｛十进制编码为32,33,35-126的ASCII字符｝"

范例：”hello world”

分析：定义了字符串，由任意个十进制编码为32，33，35-126的ASCII字符用双引号””引起来组成，双引号里面可以不出现任何字符（即””），注意ASCII编码为32的字符是空格（” ”），” ”和””不一样。

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＞

分析：定义了程序，由＜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＞组成，其中＜常量说明＞、＜变量说明＞可以不出现或者出现一次，＜有返回值函数定义＞、＜无返回值函数定义＞可以出现任意次， ＜主函数＞只能出现一次。

1. ＜常量说明＞ ::= const＜常量定义＞;{ const＜常量定义＞;}

范例：const int a=10;

分析：描述了常量说明的组成，由一个或者多个const＜常量定义＞;组成。

1. ＜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＞}

范例：int a=10,b=11

char ch=’a’

分析：描述了常量定义，可以有一个或者多个int或char类型的常量，中间由逗号分开。

1. ＜无符号整数＞ ::= ＜非零数字＞｛＜数字＞｝

范例：10

23

分析：定义了无符号整数，由一个或多个数字组成，首位不允许为0，0不属于无符号整数。

1. ＜整数＞ ::= ［＋｜－］＜无符号整数＞｜０

范例：0

-2017

分析：定义了整数，由+、-、无符号整数、0组成，符号部分+和-可以任意出现一个或者两者均不出现，数字部分为无符号整数；0为特殊情况，前面不可以有符号。

1. ＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝

范例：\_a

B1

分析：定义了标识符，由字母和数字组成，第一个字符必须为字母，后面存在任意个祖字母或者数字（包括0个的情况）。

1. ＜声明头部＞ ::= int＜标识符＞|char＜标识符＞

范例：int a

char ch

分析：定义了声明头部，由int 或char加标识符组成。

1. ＜变量说明＞ ::= ＜变量定义＞;{＜变量定义＞;}

范例：int a, b; char string[10];

分析：定义了变量说明，由一个或者多个＜变量定义＞和分号;组成。

1. ＜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＞‘]’) }

范例：int a,b

char string[10]

分析：说明了变量定义的组成，由类型标识符和＜标识符＞或＜标识符＞‘[’＜无符号整数＞‘]’组成，若出现多个＜标识符＞或＜标识符＞‘[’＜无符号整数＞‘]’，中间用逗号隔开。

1. ＜类型标识符＞ ::= int | char

范例：int

分析：定义了类型标识符，分为int和char两种。

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

范例：int foo(int a, char ch){a = 10; b = 15}

分析：定义了＜有返回值函数定义＞，由＜声明头部＞‘(’＜参数表＞‘)’‘{’＜复合语句＞‘}’组成。

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

范例：void foo(int a, char ch){a = 10; ch = ‘z’}

分析：定义了＜无返回值函数定义＞，由void＜标识符＞‘(’＜参数表＞‘)’‘{’＜复合语句＞‘}’组成。

1. ＜复合语句＞ ::= ［＜常量说明＞］［＜变量说明＞］＜语句列＞

范例：int a, b;

a = 10; b = a + 1;

分析：定义了＜复合语句＞，由＜常量说明＞、＜变量说明＞、＜语句列＞组成，＜常量说明＞和＜变量说明＞可以出现一次或者不出现，＜语句列＞只能出现一次。

1. ＜参数表＞ ::= ＜类型标识符＞＜标识符＞{,＜类型标识符＞＜标识符＞}|＜空＞

范例：int a

int a, char b

分析：定义了参数表，由任意个＜类型标识符＞＜标识符＞组成，允许为空（即不出现），若出现多个，中间由逗号隔开。

1. ＜主函数＞ ::= void main‘(’‘)’ ‘{’＜复合语句＞‘}’

分析：说明了主函数由void main‘(’‘)’ ‘{’＜复合语句＞‘}’组成。

1. ＜表达式＞ ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞}

范例：a+a\*b/(c+d)

分析：定义了表达式，由［＋｜－］＜项＞和任意个＜加法运算符＞＜项＞组成，第一个项之前的符号可以为+或-或不存在。

1. ＜项＞ ::= ＜因子＞{＜乘法运算符＞＜因子＞}

范例：a\*b/(c+d)

分析：定义了项，由＜因子＞和任意个＜乘法运算符＞＜因子＞组成。

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

范例：a

分析：定义了因子，由＜标识符＞或＜标识符＞‘[’＜表达式＞‘]’或＜整数＞或＜字符＞或＜有返回值函数调用语句＞或‘(’＜表达式＞‘)’组成，不允许同时出现或者为空。

1. ＜语句＞ ::= ＜条件语句＞｜＜循环语句＞｜<情况语句>|‘{’＜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＞;

范例：{a = 2; b = 3;}

分析：定义了＜语句＞，由＜条件语句＞｜＜循环语句＞｜<情况语句>|‘{’＜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＞;组成。允许为空。

1. ＜赋值语句＞ ::= ＜标识符＞＝＜表达式＞|＜标识符＞‘[’＜表达式＞‘]’=＜表达式＞

范例：a = a+b

Array[1] = c

分析：定义了＜赋值语句＞，由＜标识符＞＝＜表达式＞或＜标识符＞‘[’＜表达式＞‘]’=＜表达式＞组成。

1. ＜条件语句＞ ::= if ‘(’＜条件＞‘)’＜语句＞

范例：if(a>b) a=b;

分析：定义了＜条件语句＞，由if ‘(’＜条件＞‘)’＜语句＞组成。

1. ＜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

范例：a+b == 100

分析：定义了＜条件＞，由＜表达式＞＜关系运算符＞＜表达式＞或＜表达式＞组成。单独的＜表达式＞的含义是表达式为0条件为假，否则为真。

1. ＜循环语句＞ ::= do＜语句＞while ‘(’＜条件＞‘)’

范例：do a=a+1;while(a<b)

分析：定义了＜循环语句＞，由do＜语句＞while ‘(’＜条件＞‘)’组成。

1. ＜常量＞ ::= ＜整数＞|＜字符＞

范例：100

‘a’

分析：定义了常量，由整数或者字符组成，两者只能出现一个，不允许为空。

1. ＜情况语句＞ ::= switch ‘(’＜表达式＞‘)’ ‘{’＜情况表＞ ‘}’

范例：switch (a){case 1: b=2017; case 2: b=2018; case 3: b=2019 }

分析：定义了＜情况语句＞，由switch‘(’＜表达式＞‘)’‘{’＜情况表＞ ‘}’组成。

1. ＜情况表＞ ::= ＜情况子语句＞{＜情况子语句＞}

范例：case 1: b=2017; case 2: b=2018; case 3: b=2019

分析：定义了＜情况表＞，由一个或更多个＜情况子语句＞组成。

1. ＜情况子语句＞ ::= case＜常量＞：＜语句＞

范例：case 1: b=2017;

分析：定义了＜情况子语句＞，由case＜常量＞：＜语句＞组成。

1. ＜有返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’

范例：foo(a,b)

分析：定义了＜有返回值函数调用语句＞，由＜标识符＞‘(’＜值参数表＞‘)’组成。

1. ＜无返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’

范例：foo(a,b)

分析：定义了＜有返回值函数调用语句＞，由＜标识符＞‘(’＜值参数表＞‘)’组成。

1. ＜值参数表＞ ::= ＜表达式＞{,＜表达式＞}｜＜空＞

范例：a,b

分析：定义了＜值参数表＞，由若干个＜表达式＞组成，可以为0个；若出现多个表达式，中间用逗号分隔。

1. ＜语句列＞ ::=｛＜语句＞｝

范例：a = 1; b = 2;

分析：定义了＜语句列＞，由若干个＜语句＞组成，允许为0个（即空）的情况。

1. ＜读语句＞ ::= scanf ‘(’＜标识符＞{,＜标识符＞}‘)’

范例：scanf (a, b)

分析：定义了读语句，由scanf()以及括号里面的一个或多个标识符组成，标识符若有多个，用逗号分隔。

1. ＜写语句＞ ::= printf‘(’＜字符串＞,＜表达式＞‘)’|printf ‘(’＜字符串＞‘)’|printf ‘(’＜表达式＞‘)’

范例：printf(“temp”,a)

分析：定义了＜写语句＞，由printf‘(’＜字符串＞,＜表达式＞‘)’或printf ‘(’＜字符串＞‘)’或printf ‘(’＜表达式＞‘)’组成。

1. ＜返回语句＞ ::= return[‘(’＜表达式＞‘)’]

范例：return (a)

分析：定义了＜返回语句＞由return和‘(’＜表达式＞‘)’组成，‘(’＜表达式＞‘)’可以不出现，即只有单个return的情况。

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）标识符不区分大小写字母

（3）写语句中的字符串原样输出

（4）数组的下标从0开始

（5）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

注意点：

1.字母包含下划线\_

2.字符用单引号’ ’括起来

3.字符串用双引号” ”括起来，字符串双引号里面可以为空。

4.无符号整数首位不允许为0，且0不属于无符号整数。

5.整数中包含0，但是0之前不允许有符号。

6. 参数表：int a, char b

值参数表：a, b

7. 语句：单个，可以为空

语句列：多个语句

复合语句：语句列之前加上常量说明或者变量说明

8. 情况子语句：case 1: a = 10;

情况表：多个情况子语句

情况语句：情况表加上switch，此文法不包含default

### 2．目标代码说明

目标代码为MIPS指令代码，具体详见第二部分。

### 3. 优化方案\*

1.基本块内部的公共子表达式删除（DAG图）

2.数据流分析（通过活跃变量分析建立冲突图）

3.全局寄存器分配（着色算法）

## 二．详细设计

### 1．程序结构

出错处理

词义分析

语法分析

词法分析

生成汇编码

生成四元式

0

符号表管理

### 2．类/方法/函数功能

词法分析：

|  |  |
| --- | --- |
| 函数 | 功能 |
| bool isSpace() | 判断读进的字符是否为空格 |
| bool isNewLine() | 判断读进的字符是否为换行 |
| bool isTab() | 判断读进的字符是否为制表符 |
| bool isDigit() | 判断读进的字符是否为数字 |
| bool isLetter() | 判断读进的字符是否为字母 |
| void clearToken() | 清空token |
| void catToken() | 将读到的字符拼接进token |
| void toLow() | 大写字母转换为小写 |
| int reserver() | 判断token中的内容是否为保留字 |
| int transNUM() | 将token中的字符串转换为数字 |
| void getsym() | 词法分析，获取一个单词 |

语法分析：

|  |  |
| --- | --- |
| 函数 | 功能 |
| void const\_def() | 分析常量定义 |
| void const\_statement() | 分析常量声明 |
| void var\_def() | 分析变量定义 |
| void var\_statement() | 分析变量声明 |
| void expr() | 分析表达式 |
| void term() | 分析项 |
| void factor() | 分析因子 |
| void returnable\_statement() | 分析有返回值函数声明 |
| void void\_statement() | 分析无返回值函数声明 |
| void parameter\_list() | 分析参数表 |
| void compound\_statement() | 分析复合语句 |
| void main\_statement() | 分析主函数 |
| void assign\_statement() | 分析赋值语句 |
| void condition\_statement() | 分析条件语句 |
| void condition() | 分析条件 |
| void do\_while\_statement() | 分析循环语句 |
| void switch\_statement() | 分析情况语句 |
| void case\_list(string caselabel) | 分析情况表 |
| void case\_statement(string caselabel) | 分析情况子语句 |
| void call\_returnable\_statement() | 分析有返回值函数调用语句 |
| void call\_void\_statement() | 分析无返回值函数调用语句 |
| void value\_table(string funcname) | 分析值参数表 |
| void statement\_list() | 分析语句列 |
| void statement() | 分析语句 |
| void read\_statement() | 分析读语句 |
| void write\_statement() | 分析写语句 |
| void return\_statement() | 分析返回语句 |
| void program() | 分析整个程序 |

符号表：

|  |  |
| --- | --- |
| 函数 | 功能 |
| void insert\_table() | 插入符号表 |
| bool find\_table() | 查表 |

四元式：

|  |  |
| --- | --- |
| 函数 | 功能 |
| void insert\_four() | 生成新的四元式 |
| string get\_temp() | 获得临时变量 |
| void clear\_temp() | 删除所有的临时变量 |
| string new\_label() | 获得label |
| void set\_label() | 设置label |

生成目标代码：

|  |  |
| --- | --- |
| 函数 | 功能 |
| void golbal() | 生成MIPS .data内容 |
| void push\_mips() | 参数转MIPS |
| void call\_mips() | 函数调用四元式转MIPS |
| void add\_mips() | 加法运算四元式转MIPS |
| void sub\_mips() | 减法运算四元式转MIPS |
| void mult\_mips() | 乘法运算四元式转MIPS |
| void div\_mips() | 除法运算四元式转MIPS |
| void assign\_mips() | 赋值语句四元式转MIPS |
| void return\_mips() | 返回语句四元式转MIPS |
| void scanf\_mips() | 读语句四元式转MIPS |
| void printf\_mips() | 写语句四元式转MIPS |
| void branch\_mips(int i) | 分支语句四元式转MIPS |
| void assign\_array\_mips() | 数组赋值语句四元式转MIPS |
| void use\_array\_mips() | 数组使用语句四元式转MIPS |
| void to\_mips() | 四元式生成MIPS |

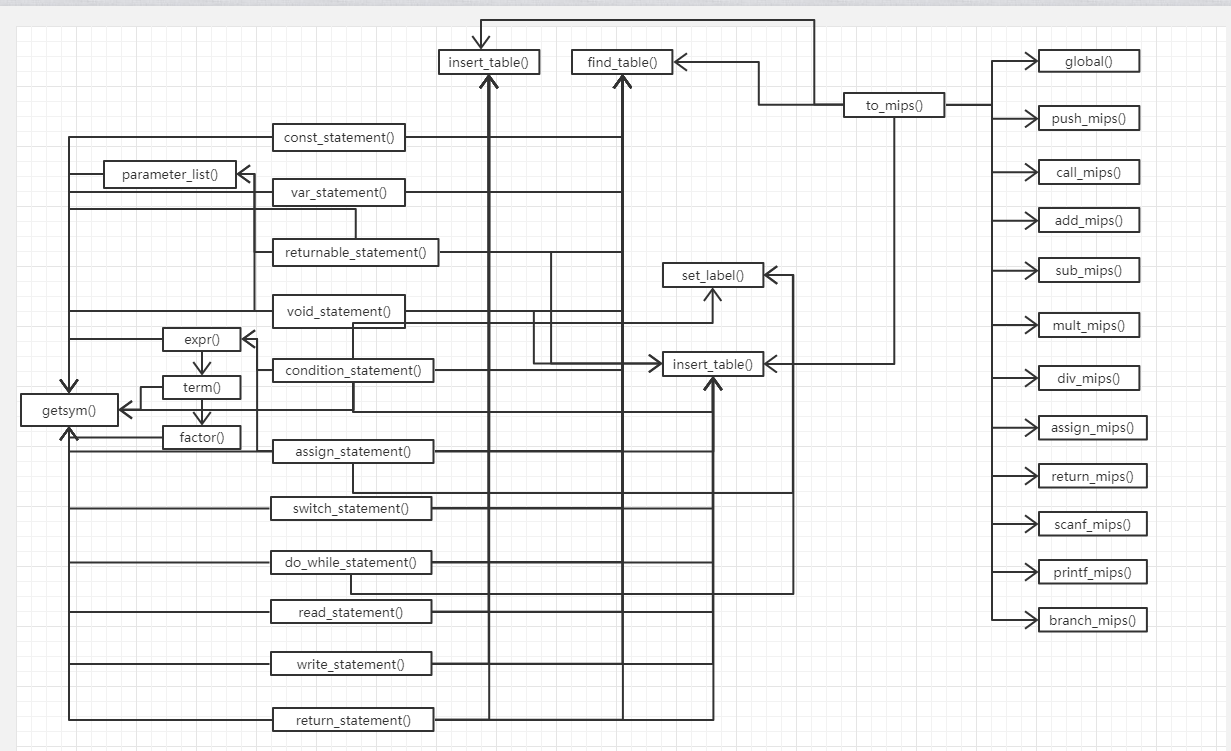
错误处理：

|  |  |
| --- | --- |
| 函数 | 功能 |
| void error() | 打印错误信息，并跳转到下一个合法字符 |

优化：

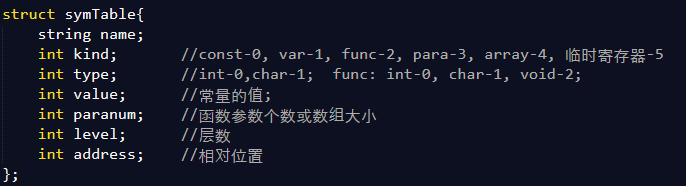
|  |  |
| --- | --- |
| 函数 | 功能 |
| void dag() | DAG图消除局部公共子表达式 |
| void active\_var() | 活跃变量分析，生成冲突图 |
| void color\_graph() | 着色算法分配全局寄存器 |

### 3．调用依赖关系



### 4．符号表管理方案

采用无序表的存储方式，具体数据结构如下：



### 5．存储分配方案

优化前：

所有常量、变量、临时变量的值均存储在运行栈中，每次使用前从运行栈中load，使用结束后save进去。

优化后：

根据全局寄存器的分配，将部分值存储在寄存器中，其余值存储在运行栈中，减少程序运行时间。

运行栈结构如下：（和课本略有不同）

调用新函数

出现函数调用

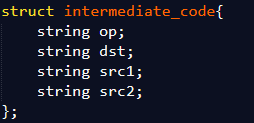
|  |
| --- |
| 局部常量变量 |
| 局部临时变量 |
| 实参 |
| $sp |
| $fp |
| $ra |
| 参数 |
| 局部常量变量 |
| …… |
| 以下内容同上 |

### 6. 解释执行程序\*

本程序不采用解释执行程序，而是编译执行。

### 7. 四元式设计\*

四元式结构如下：



|  |  |
| --- | --- |
| 操作种类 | 四元式设计 |
| + - \* / | op dst src1 src2 |
| >= > <= < == != | op dst src1 |
| 赋值 | = dst src1 |
| 数组赋值 | []= dst index src2 |
| 数组使用 | array dst arrayname index |
| 函数调用 | call funcname paranum |
| 函数实参 | push paravalue |
| 函数声明 | int/char/void funcname () |
| 函数形参 | para dst |
| 输出 | printf [string] expr |
| 输入 | scanf dst |
| 返回 | return dst |
| 分支 | label  BZ/BNZ label |
| 函数结束 | end funcname |
| Switch结束 | end switch |

### 8. 目标代码生成方案\*

1. 在.data中为全局常量，变量分配空间

2. 遍历四元式找到需要输出的字符串，并且给它命名，然后存储在.data中

3. 添加main函数的入口，j main;

4. 根据每条四元式选择合适的mips指令翻译，注意四元式中的显式操作数在mips中需要从内存或者寄存器中的正确位置寻找

### 9. 优化方案\*

1.基本块内部的公共子表达式删除（DAG图）

2.数据流分析（通过活跃变量分析建立冲突图）

3.全局寄存器分配（着色算法）

### 10. 出错处理

void error(int errflag){

1：不符合常量定义格式

2：不符合变量定义格式

3：函数结构错误

4：不符合参数表格式

5：不能给常量赋值

6：标识符未定义

7：不能给数组整体直接赋值

8：标识符重复定义

10：调用的函数不存在

}

其余错误处理在其他函数内部处理，并没有写入error函数中，包括：

1. 缺少小括号

2. 缺少中括号

3. 缺少大括号

4. 缺少分号

5. 非法字符

6. 函数return内容和函数类型不符

7. 函数形参实参个数不相等

8. 函数形参实参类型不匹配

9. 常量定义、变量定义右侧类型错误

10. 数组越界

11. 函数名冲突

12. 参数名冲突

13. 标识符不是数组

14. 不符合赋值语句结构

15. 条件不符合要求

16. 不符合循环语句结构

17. case后面不是整数或者字符

18. 未知语句

19. scanf中内容不能为非变量

20. printf中的内容不符合要求

## 三．操作说明

### 1．运行环境

运行环境为64位win10系统，运行软件为Visual Studio 2010版本。

### 2．操作步骤

程序运行之后在控制台输入完整的测试程序路径

错误信息直接在控制台输出

E:/result.txt输出词法分析结果

E:/statement.txt输出语法分析结果

E:/four.txt输出四元式结果

E:/mips.txt输出最终生成的目标代码

优化后：

E:/four2.txt输出消除公共子表达式的四元式

E:/mips2.txt输出全局寄存器分配之后的目标代码

## 四．测试报告

**1．测试程序及测试结果**

**1.txt（正确）**

输入： 6

12

x

2

输出：

testDefine passed!

Please input a num: 6

The result of jiecheng is: 720

If test passed!

Please input a positive int: 12

The result of fibo is: 144

input z or x: x

world

Switch test passed!

a

b

Print character test passed!

input 1~5: 2

4

9

16

25

与预期结果相符合。

**2.txt（正确）**

输入：

3

输出：

please input 1~4: 3

not equal 1

equal 3

a

b

c

d

e

与预期结果相符合。

**3.txt（正确）**

输入：

3

a

输出：

please input a int(1~3): 3

please input a char(a~d): a

33333333

aaaaaa

与预期结果相符合。

**4.txt（正确）**

输入：

3

3

2

8

4

3

5

9

12

输出：

---------pow\_test---------

input bottom: 3

input exponent: 3

pow: 27

input bottom: 2

input exponent: 8

pow: 256

input bottom: 4

input exponent: 3

pow: 64

---------pow\_test---------

---------fibo\_test---------

input: 5

fibo: 5

input: 9

fibo: 34

input: 12

fibo: 144

---------fibo\_test---------

与预期结果相符合。

**5.txt（正确）**

输出：

testDefine passed!

z

121

y

122

121

与预期结果相符合。

**6.txt（错误）**

输出：

1 num1 常量定义右侧类型错误

2 error! without semi!

2 ch\_1 常量定义右侧类型错误

41 error! 调用的函数不存在

84 error! 标识符未定义

85 error! 标识符未定义

86 error! 标识符未定义

86 error! 没有)

**7.txt（错误）**

输出：

7 error! 不能给数组整体直接赋值

28 error! 未知语句

33 error! 没有)

34 error! 函数调用形参实参类型不符

34 error! 函数调用参数个数不正确

48 error! 无返回值函数有返回值

**8.txt（错误）**

输出：

16 error! 标识符未定义

21 error! case后面不是整数或者字符

21 error! 缺少：

21 error! 没有}

22 error! 未知语句

**9.txt（错误）**

输出：

9 error! 有返回值函数无返回值

23 error! 函数调用形参实参类型不符

34 error! 函数调用参数个数不正确

34 error! 没有分号

39 error! 标识符未定义

46 error! 标识符未定义

47 error! 标识符未定义

47 error! 条件不符合要求

47 error! 没有)

52 error! 调用的函数不存在

**10.txt（错误）**

输出：

2 error! 标识符重复定义

3 int\_0 常量定义右侧类型错误

5 char0 常量定义右侧类型错误

6 error! 标识符重复定义

9 error! 不符合变量定义格式

14 error! 不符合变量定义格式

19 error! scanf中的内容不能为数组整体

21 error! 没有)

**2．测试结果分析**

**1.txt（正确）**

本程序重点测试了常量声明，变量声明，函数调用，递归，输入输出语句，赋值语句，返回语句，条件语句，循环语句，情况语句，表达式。

程序的功能是输入三个整数和一个字符，输出斐波那契值，阶乘，字符串等。

**2.txt（正确）**

本程序重点测试了条件语句，循环语句和数组的使用，同时还有常量变量声明，输入输出语句，返回语句，赋值语句等。

程序的功能是输入一个数字，判断它的结果，并输出相应的提示；通过循环存储并输出数组中的内容。

**3.txt（正确）**

本程序重点测试了情况语句，同时还有变量声明，输入输出语句，返回语句。

程序的功能是输入一个整数和字符，分别根据整数和字符的值，输出对应的情况语句中的提示。

**4.txt（正确）**

本程序重点测试了函数的调用和递归，同时华友变量定义，条件语句，循环语句，输入输出语句，赋值语句，表达式的运算，返回语句等。

程序的功能是根据提示输入一组数字，输出其对应的指数运算的值或者斐波那契值。

**5.txt（正确）**

本程序重点测试了常量变量的声明，输出语句，表达式的运算，同时还有赋值语句等等。

程序的功能是对int和char类型的变量，进行交叉的表达式的运算，输出对应的值。

**6.txt（错误）**

本程序重点测试了常量定义类型检查，没有分号，调用函数不存在，标识符未定义，没有右括号等错误。

其中line 1：int 类型的num1右侧为char

line 2：char 类型的ch\_1右侧为int

line 2: 最后缺少分号

line 41：调用testDefine1()；该函数不存在，应该为testDefine()；

line 84 85 86 使用了未定义的ch1；

**7.txt（错误）**

本程序重点测试了不能给数组整体赋值，未知语句，函数调用形参实参类型不符，函数调用参数个数不正确，无返回值函数有返回值。

其中line 7：给数组ch3整体赋值

Line 28：else 为未知语句

Line 33： scanf语句缺少右括号

Line 34：函数调用参数个数不正确

Line 48：无返回值函数有返回值

**8.txt（错误）**

本程序重点测试了case后面不是整数或者字符，未知语句，标识符未定义。

其中：line 16：标识符b未定义

Line 21：case后面不是整数或者字符

Line 22：default 为未知语句

**9.txt（错误）**

本程序重点测试了有返回值函数无返回值，函数调用形参实参类型不符，函数调用参数个数不正确，没有分号，标识符未定义，调用的函数不存在等。

其中line 9：有返回值函数无返回值

Line 23：pow函数调用形参实参类型不符

Line 34：函数调用参数个数不正确，没有分号

Line 39 46 47：标识符未定义

**10.txt（错误）**

本程序重点测试了标识符重复定义，常量定义右侧类型错误，不符合变量定义格式，scanf中的内容不能为数组整体，没有)

其中line 2：int0标识符重复定义

Line 3: int\_0 常量定义右侧类型错误

Line 5: char0 常量定义右侧类型错误

Line 6: var0标识符重复定义

Line 19: scanf中array为数组整体

Line 21: 没有)

## 五．总结感想

不知不觉，编译课程设计已经接近尾声了，在这三个月里，我学到了很多，有编译知识方面的，有编程框架设计方面的，还有学习习惯和学习精神方面的。

实验最初开始的时候，我一直在考虑选难度三还是难度二，后来我思考了一下，编译实验应该是本科阶段最后一个代码量很大的作业了，我还是想挑战一下自己，不要留下遗憾，所以最终我选择了难度三。

前几周还是比较轻松的，词法分析按照书上给的算法容易就能实现，语法分析代码量很大，但是逻辑也比较简单。简单轻松的前几周给了我一种编译其实也不是很难的错觉，然后就到了生成目标代码那一周了，在一周里实现了符号表管理，四元式生成，目标代码生成，工作量非常大。我也很后悔之前几周没有稍微写一点后面的内容，导致那一周通宵了两天。目标代码生成结束之后，我又增加了比较详细的错误处理，然后再陆陆续续进行了一些优化，总之，这三个月过得还是比较充实的。

编译器写完之后，我有很多编程方面的感想。首先，在开始一个项目之前，不要着急先动手开始写，要先思考好整体框架，要写哪几部分的内容，每一部分分别要实现怎么样的功能，各部分之前怎样联系起来。然后，在项目开始之后要进行合理的编辑，一个很大的工程最好不要写在同一个文件里面，这次的编译实验，我把所有的代码都写在了一个.c文件里面，最后发现有五六千行，找起函数来非常不方便，直观看上去也不是很清晰。还有就是，变量命名的问题，因为之前没有好好命名，导致我后来在代码中添加一些内容的时候，不知道每个变量的含义，常常需要花大量的时间去研究。如此种种，都使得我的编程效率降低了很多，这次编译实验使我的一些很不好的编程习惯都暴露出来了，这也有利于我以后的进步。

总之，编译课设让我感受到了良好的编程习惯的重要性，也让我明白了不管是学习还是生活都要先有一个明确的规划，然后按照计划一步一步推进，最终一定能够实现目标。