

**编译原理课程设计**

编译器技术文档

北京航空航天大学

计算机学院

陈麒先

16061160

二○一九年一月

**郑重声明**

**关于诚实守信公约：**

**本技术文档中参考了文献或互联网资料的部分均有引用标注，另有与老师或同学研究后的成果引用将在特别致谢中说明，其余部分均为本人通过查阅课件、课堂笔记和教材后独立思考的结果。另，技术文档撰写仓促，如有错误，在所难免，欢迎批评指正！特此声明。**

**陈麒先**

**原创性声明**

**本技术文档中未标注部分内容由作者原创。抄袭行为在任何情况下都是不能容忍的(COPY is strictly prohibited under any circumstances)！转载或引用须征得作者本人同意，并注明出处！勿谓言之不预！**

**陈麒先**



**目录**

[一、需求分析 1](#_Toc534036103)

[1．文法说明 1](#_Toc534036104)

[2．目标代码说明 9](#_Toc534036105)

[3. 优化方案 10](#_Toc534036106)

[二．详细设计 12](#_Toc534036107)

[1．程序结构 12](#_Toc534036108)

[2．类/方法/函数功能 12](#_Toc534036109)

[3．调用依赖关系 18](#_Toc534036110)

[4．符号表管理方案 18](#_Toc534036111)

[5．存储分配方案 19](#_Toc534036112)

[6. 解释执行程序 20](#_Toc534036113)

[7. 四元式设计 20](#_Toc534036114)

[8. 目标代码生成方案 22](#_Toc534036115)

[9. 优化方案 23](#_Toc534036116)

[10. 出错处理 23](#_Toc534036117)

[三．操作说明 27](#_Toc534036118)

[1．运行环境 27](#_Toc534036119)

[2．操作步骤 27](#_Toc534036120)

[四．测试报告 28](#_Toc534036121)

[1．测试程序及测试结果 28](#_Toc534036122)

[2．测试结果分析 28](#_Toc534036123)

[五．总结感想 37](#_Toc534036124)

编译器技术文档

一、需求分析

## 1．文法说明

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＞

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

＜变量说明＞ ::= ＜变量定义＞;{＜变量定义＞;}

＜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＞'}'

＜主函数＞ ::= void main'('')''{'＜复合语句＞'}'

【范例】

const int a = 1; // 常量说明

int b; // 变量说明

int add(int x,int y){ //有返回值函数定义

return x + y;

}

void print(){ //无返回值函数定义

printf("helloworld");

}

void main(){ //主函数

int c;

b = 1;

c = add(a,b);

print();

}

【分析】

- 根据程序的文法可知，程序各个组成成分的顺序已经被限定好了，不能随便更改顺序，如“int a,b; const x=10;”此种顺序是不允许的。

- 每一个程序按顺序由常量说明部分，变量说明部分，有无返回值函数定义部分，以及主函数组成。其中常量，变量，过程及函数说明说明，复合语句之间有严格的先后顺序，不能打乱。而相连的有无返回值的函数说明部分则可以随意改变顺序。变量说明，常量说明，过程及函数说明对于一个分程序来说均为可有可无的部分，有无返回值的函数定义可以有零到多个。只有最后的主函数是必须存在的。

2.

＜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＞}

＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝

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

＜无符号整数＞ ::= ＜非零数字＞｛＜数字＞｝| 0

＜整数＞ ::= ［＋｜－］＜无符号整数＞

＜数字＞ ::= ０｜＜非零数字＞

＜非零数字＞ ::= １｜．．．｜９

＜字母＞ ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

＜加法运算符＞ ::= +｜-

＜乘法运算符＞ ::= \*｜/

【范例】

const int \_aA = 0; //int常量

const char Ch = '+'; //char常量

【分析】

- 常量说明部分可以说明一至多个常量，每条定义必须有const开始，以分号结束，中间是常量定义。

- 常量定义可以是int也可以是char，对统一类型可定义多个常量，以逗号分隔，定义由等号连接，左边必须为标识符。

- 标识符必须以字母开头，有字母数字组成。

- 字母包含大小写和下划线，程序对大小写敏感。

- 整数的开头为可选的+-加上无符号整数。

- 无符号整数可以是0或者不允许前0的其他数字。

- 字符是能是+-\*/\_大小写字母和数字。

3.

＜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＞']' )}

＜类型标识符＞ ::= int | char

【范例】

int a; // 声明int变量

char b[100]; // 声明长100字符数组

【分析】

- 变量定义不支持在定义时直接赋值，如不允许‘int a = 1;’。

- 类型标识符包括2种，int和char。

- 变量定义包括变量和数组定义,数组定义中无符号整数值表示数组长度，值大于0。

- char类型的变量或常量，用字符的ASCII码对应的整数参加运算。

4.

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

＜声明头部＞ ::= int＜标识符＞ |char＜标识符＞

＜参数表＞ ::= ＜参数＞{,＜参数＞}| ＜空>

＜参数＞ ::= ＜类型标识符＞＜标识符＞

＜复合语句＞ ::= ［＜常量说明＞］［＜变量说明＞］＜语句列＞

【范例】

int add\_1(int x,int y){ //声明头部 参数表 复合语句

const int a = 1; // 常量说明

int result; // 变量说明

result = x + y + a; // 语句列

return result;

}

【分析】

- 有返回值函数定义有声明头部（参数表）{复合语句}组成，顺序不得更改，三者不可缺省，（）{}为必有字符。

- 声明头部由int,char+标识符组成，表示声明的函数返回值类型，二者只能选择其一进行声明。

- 参数表由零至多个参数，传入参数可以为空。各个参数之间由逗号分隔。

- 参数为一个类型标识符加一个标识符组成，不允许多个标识符的情况出现在参数中，如不允许‘int a,b’。

- 复合语句由常量、变量说明和语句列组成，顺序不可调换，常量、变量为可选项，必须有语句列。

- 有返回值的函数定义文法上允许无返回值，但是不符合语义需要报错。

5.

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

【范例】

void hello(){ //标识符 参数表 复合语句

printf("hello world!");

}

【分析】

- 无返回值即void 标识符为函数名，其余部分均与有返回值函数定义相同。

- 此处范例给出了参数表空的情况。

- return();是不合法的。

6.

＜主函数＞ ::= void main'('')''{'＜复合语句＞'}'

【范例】

void main(){

printf("hello world!");

}

【分析】

- 主函数中的各个成分是确定的void main(){ 中间为复合语句块 最后以}结尾。

- 无返回值的函数可以有return语句，有返回值可以没有return语句。

7.

＜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＞

【范例】

① a + b

② a[i] \* b + (c[j + k] + d \* e)

③ a \* 4 - 'a'/add(x,y)

④ - a[i \* 4 + base] + c/(d \* e[j])

【分析】

- 表达式有一至多个项组成，每个项之间有加法运算符链接，[+|-]只作用于第一个项，即第一项之前也可有加法算符。

- 项由一至多个因子组成，因子之间由乘法运算符连接。

- 因子可以是一个标识符，即常量或者变量；或是＜标识符＞'['＜表达式＞']'，即一个数组，数组下标为【】中的内容，也可以通过一个表达式进行计算；或是'('＜表达式＞')'由（）包含的部分整体为一个表达式的求值语句；或是一个带返回值的函数调用，通过函数获取数值；或是整数、字符。综上所述，因子应该得到一个数值带回到项中进行计算。

8.

＜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＞;

【范例】

有关语句的范例将通过每个语句语法成分分别给出，敬请期待。

【分析】

- 语句的定义比较易于理解，即是各种不同的语句类型的集合。其中要注意有分号结尾的情况，除了条件、循环、情况语句和'{'＜语句列＞'}'之外，都有分号结尾。注意空语句直接只有一个分号。

9.

＜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＞

＜关系运算符＞ ::= <｜<=｜>｜>=｜!=｜==

【范例】

if(a == b){

printf("hello world");

}

【分析】

- 条件语句由关键字if引导，根据（）中的条件进行跳转，若条件为真则执行语句否则不执行。

- 条件分两种，一种是由有关系运算符连接的两个表达式形成的判断条件，另一种是单表达式形成的判断条件。由前文分析可知，表达式会返回数值，表达式为0条件为假，否则为真；而由关系运算符连接的两个数值运算结果将返回0、1，其中0为假，1为真。

- 关系运算符为数值比较运算，返回真值。

10.

＜循环语句＞ ::= while '('＜条件＞')'＜语句＞

【范例】

while(i < n){

printf(i);

i = i + 1;

}

【分析】

- 关键字while ，条件为上文分析过的，如果条件值为真，则执行语句，直到不满足条件退出循环。

11.

＜语句列＞ ::= ｛＜语句＞｝

【范例】

a = 1 + 1;

b[9] = a;

c = b[9] \* 6;

【分析】

- 语句列即一组语句的序列集合，由零到多个语句构成。

12.

＜有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＞

【范例】

add(x,y)

【分析】

- 注意有返回值函数调用语句在语句定义中，其应有的分号在语句定义中体现。

- 注意各个成分不得缺省和调换顺序，标识符为调用的函数名。

- 值参数表为传入的实参列表，必须与函数定义的形参相对应，每个相应位置都应该出现数值，可通过表达式求得。

13.

＜无返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'

【范例】

print()

【分析】

- 比有返回值的函数调用简单，标识符为函数名，由（）包含参数列表，这里的范例给出值参数表为空的情况。

- 同样分号是定义在语句中的。

14.

＜赋值语句＞ ::= ＜标识符＞＝＜表达式＞|＜标识符＞'['＜表达式＞']'=＜表达式＞

【范例】

① a = 1

② b[0] = 1

③ c[i \* 4 + base] = 1

【分析】

- 赋值可以为变量或数组中的某一个元素赋值，有=连接。

- 所有形如＜标识符＞'['＜表达式＞']'均表示一个数组，表达式中计算数组索引值，数组的下标从0开始。

- 赋值语句等号右边不能是比较表达式。

15.

＜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＞')']

【范例】

① scanf(a)

② printf("hello world",a + b)

③ return （a + b）;

【分析】

- 读写语句关键词和（，）不可缺少，按顺序排列。标识符为变量名或数组名。

- 写语句分三种，可以写字符串也可写表达式，也可同时写。字符输出char，其余情况输出整数。不能printf(),字符串中不允许双引号（34号字符）。

- 返回语句由关键字return起始，后边的返回值为可选项。对于无返回值的函数定义直接return即可，若有返回值则必须在表达式外加上（）。

- 写语句中，字符串原样输出，单个char类型的变量或常量输出字符，其他表达式按整型输出，写语句自带换行。

- 读语句和变量类型有关，char读入字符，int读整数或字符的ascii码。

16.

＜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＞

【范例】

switch(state){

case 0:{ // 情况表

a = 0;

b = 1;

}

case 1 : a = 1;

default: a = 10; // 缺省

}

【分析】

- 情况语句由关键字switch(){}引导，（）中的表达式为待判断的数量，由{}包含情况表case和缺省情况default。

- 情况表有一至多个情况子语句构成。

- 每个情况子语句由case引导，后面的常量为判断条件，若与表达式的值相等则执行冒号后面的语句。范例中给出了两种可能的语句情况。

- 缺省有default:引导，当前面没有一个case生效的时候执行default后面的语句，缺省情况可以为空。

- <空>是一个特殊的符号，他是一个终结符号ε。

- 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。

## 2．目标代码说明

生成的目标代码为MIPS汇编代码，采用MIPS-C指令集中的基本指令与扩展指令。

【R型指令】

add 加法运算

sub 减法运算

mul 乘法运算

div 除法运算

【I型指令】

addi 加立即数运算

subi 减立即数运算

lw 加载字

sw 存入字

【branch类跳转指令】

beq 相等跳转

bne 不等于跳转

bge 大于等于跳转

bgt 大于跳转

ble 小于等于跳转

blt 小于跳转

【jump类跳转指令】

j 无条件跳转

jr 跳转到寄存器中存储的地址

jal 跳转并链接，返回地址存储在$ra中，用于函数调用

【扩展指令】

li 加载立即数

move 寄存器赋值

syscall 系统调用

la 加载地址

## 3. 优化方案

【方案1】 基本块内优化

- 基本块DAG图

- 消除公共子表达式

- 从DAG图重新导出中间代码

- 少量的窥孔优化

- 常数合并及传播

【方案2】全局优化

- 引用计数法

- 数据流分析

- 活跃变量分析

- 根据分析结果构造冲突图

【方案3】其他方案

在本实验中采用了方案一种的常数合并传播算法进行了基本块内的优化。

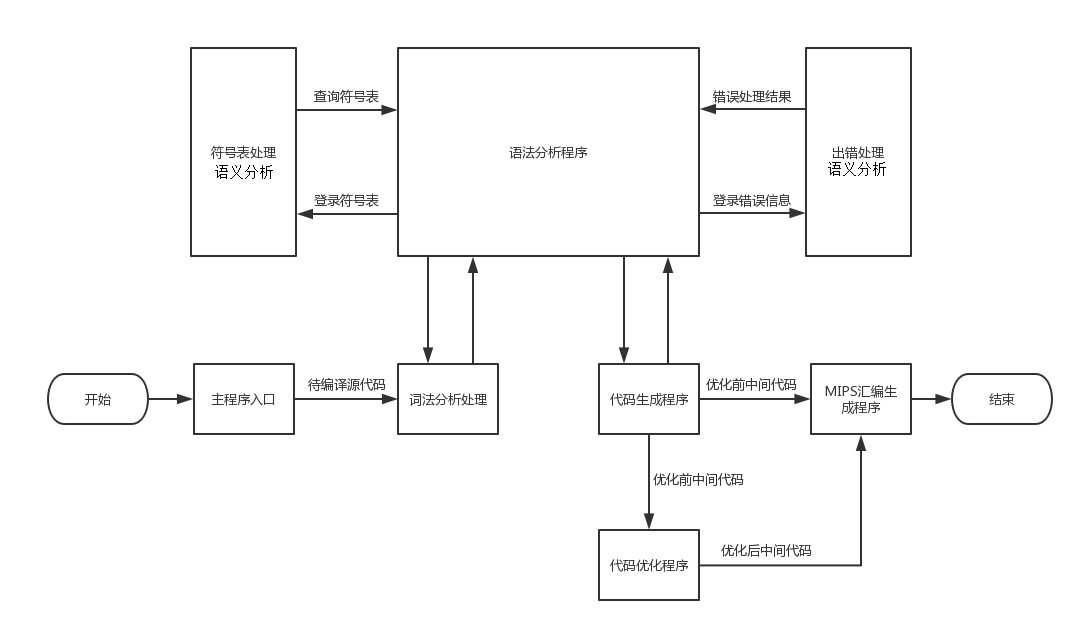
# 二．详细设计

## 1．程序结构

【概述】

程序将仿照Pascal-S程序进行编写。按照编译流程的五个阶段分阶段执行。

【结构框图】



## 2．类/方法/函数功能

编译器采用C++打造。但仅用到了C++中字符串处理的便捷特性及其数据结构，并未用到与类有关内容，属于面向过程的编程实现方案。以下对各个文件中的核心函数进行解读分析。

(1) 参数定义

【包括文件】

parameters.h ; parameters.c

【功能概述】

用于预定义编译器中所采用的数据结构和变量。

设置自定义类型、常量、全局变量等参数。

【函数定义】

(2) 编译初启

【包括文件】

main.h;main.c;initial.h ; initial.c

【功能概述】

用于编译阶段的初启动。

【函数定义】

- main.c

程序主函数入口。

完成了编译初始化、打开待写文件、启动语法分析主函数、调用代码优化模块以及调用生成中间代码模块。

反馈编译成功/报错信息。

- initial.c

init(); 用于外部调用。

readFile(); 读待编译源代码文件到code字符串数组中。

(3) 词法分析

【包括文件】

lexical.h ; lexical.c

【功能概述】

用于从源代码中获取下一符号类别至sy，符号值至id（sy,id均为全局变量）。

【函数定义】

void getsym(); // 从源代码通过反复调用上述两个函数读入下一个单词，将单词类型存储至全局变量sy，如果读到为标识符将名称存入全局变量id，如果读到数字则计算其数值，并将其计算出来将结果存储至全局变量inum，如果读到字符串则将其存储入字符串表s\_tab，如果读到比较运算符等符号则只设置sy为相应类型，其余非法字符进行报错处理

char getch(); // 用以从源代码中读取下一字符。在getsym()中被调用

void retract(); //指针回退函数。由于采取预读字符的设计，所以读到的某些分支情况需要指针回退。

void catToken(char c); // 链接到token

char catNRead(char c);// 链接并读下一字符

void singalToken(char c); // 最后一个链接

symbol reserver(char \*s); // 保留字的判断

void lexical\_output(); // 词法分析结果文件输出

(4) 语法分析&语义分析

【包括文件】

block.h ; block.c ; enter.h ; enter.c

【功能概述】

用于对待编译源代码的语法和语义进行分析。

分析其中的语法成分，以及每一个语句所属的语句类型。

将信息登录至各个表项中，并生成四元式中间代码。

进行出错处理。

【函数定义】

- grammar.c

void grammar(); 供主函数调用启动语法分析程序

void grammar\_output(); 语法分析结果输出

void constDeclare(); 常量定义处理

void varDeclare(); 变量定义处理

void addPara(); 添加参数

void paraList(); 参数列表处理

std::string factor(bool& is\_char); 因子处理

std::string item(bool& is\_char); 项处理

std::string expression(bool& is\_char); 表达式处理

void condition(); 条件处理

void ifStatement(); 条件语句处理

void whileStatement(); 循环语句处理

void paraCheck(bool is\_char); 参数检查

void valueParaList(int idx\_in\_tab); 值参数表

void voidFuncCall(); 无返回值函数调用处理

void retFuncCall(); 有返回值函数调用处理

void assignStatement(); 赋值语句处理

void scanfStatement(); 输入语句处理

void printfStatement(); 输出语句处理

void returnStatement(); 返回语句处理

void constant(); case常量

void caseStatement(); case语句处理

void caseTable(); 情况表处理

void defaultCase(); 缺省处理

void switchStatement(); 条件分支语句处理

void statement(); 语句处理

void compoundStatement(); 复合语句处理

void addFunc(); 添加函数

void funcDeclares(); 函数定义处理

void voidFuncDeclare(); 无返回值函数定义

void returnFuncDeclare(); 有返回值函数定义

void mainFuncDeclare(); 主函数定义

- enter.c

void enter\_tab(std::string idname , objtyp object , identyp idtype , int length , int adrr);

登录内容至符号表

void enter\_mid\_code(std::string z , std::string x , opration op , std::string y);

登录中间代码表

int lookup\_field(std::string name);

查当前域的符号表

int lookup\_global(std::string name);

查全局变量层符号表

int lookup\_func(std::string name);

按函数名查表

int lookup\_var(std::string name);

按变量名查表

void enter\_mips\_0(std::string labels);

void enter\_mips\_1(std::string z , std::string x , std::string y);

void enter\_mips\_2(mipsop op , std::string z , int imm);

void enter\_mips\_3(mipsop op , std::string z , std::string x);

void enter\_mips\_4(mipsop op , std::string z , std::string x , int imm);

void enter\_mips\_5(mipsop op , std::string z , std::string x , int imm); // sw $t0 0($sp)

void enter\_mips\_6(mipsop op , std::string z);

void enter\_mips\_7(mipsop op , std::string z , std::string x , std::string y);

void enter\_mips\_8(mipsop op , std::string x , std::string y , std::string labels);

按照各种目标代码格式登录目标代码表

(5) 错误处理

【包括文件】

error.h ; error.c

【功能概述】

用于处理源代码在编译过程中遇到的错误。输出报错信息。

【函数定义】

error(int errnum); 错误。根据错误类型输出报错信息和出错位置。

fatal(int fatalno); 致命错误。如超出表的存储。输出致命错误类型，强制终止编译过程。

void err\_open(); 错误信息结果输出至文件。

(6) 编译优化

【包括文件】

optimize.h ; optimize.c

【功能概述】

生成优化版本的中间代码。

【函数定义】

optimize(); 提供给外部调用，用于完成代码优化操作。

(7) 生成目标代码

【包括文件】

mid2Mips.h ; mid2Mips.c

【功能概述】

将中间四元式代码转化成MIPS汇编码。

【函数定义】

void mid2mips() 目标代码生成函数入口，供主函数调用

midoptyp midop\_analyse(std::string op , int& offset) 分析中间代码操作符种类，返回值为种类，参数修改该操作符的地址偏移，或常量数值

void gen\_mips\_str() 生成目标代码中.data域的字符串

void loadOp(std::string op , std::string reg) 根据op加载值到寄存器

void storeOp(std::string op , std::string reg) 根据op将寄存器值写入内存相应位置

identyp getType(std::string op) 获取操作符数据类型

void readOp() 处理read操作

void writeOp() 处理write操作

void funcOp() 处理函数标签

void funcendOp() 处理函数返回

void pushOp() 处理参数压栈

void callOp() 处理函数调用

void calculationOp() 处理计算指令

void assignOp() 处理赋值指令

void branchOp() 处理branch类跳转指令

void retOp() 处理返回操作

(8) 其他函数功能描述

【包括文件】

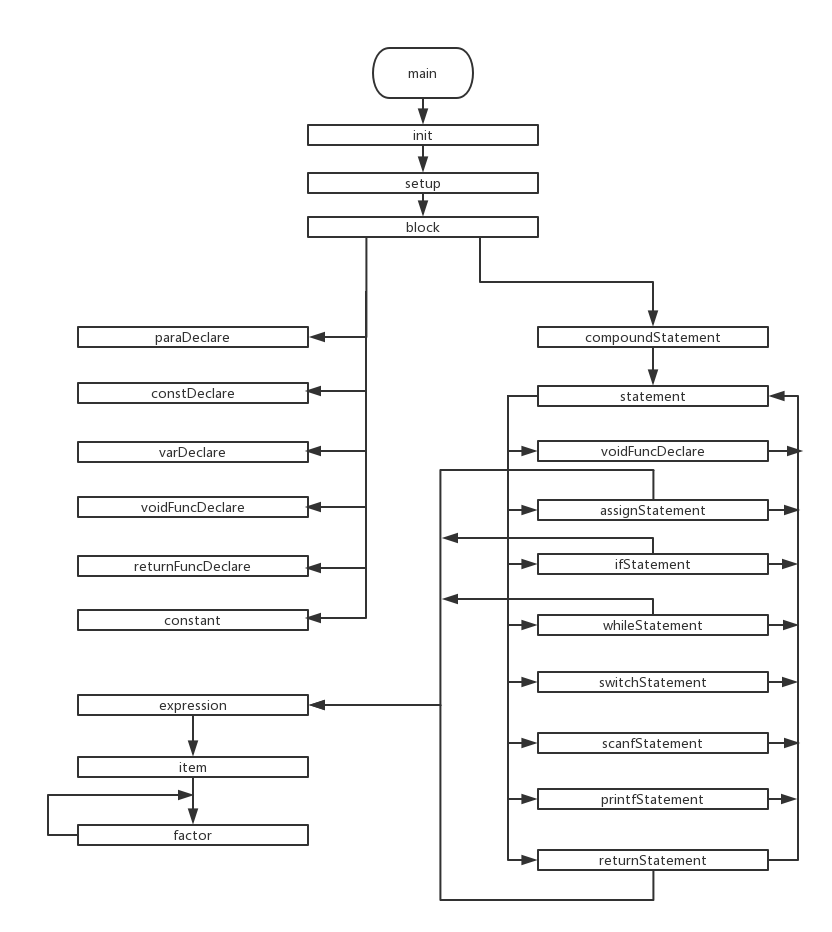
functional.h;functional.c

【功能概述】

为程序提供附属功能函数组件支持。

## 3．调用依赖关系

【调用依赖关系图】



## 4．符号表管理方案

【概述】

符号表均采用struct结构体进行存储。

采用简化的符号表结构，使用最简单的数据类型，建立二级数组索引表。

采用无序符号表实现。

【具体实现】

1. tab 符号表

【索引变量】tabidx

struct tab\_entry{

name; 标识符名

obj; 标识符种类：常量，变量，函数 , 数组 , 参数

typ; 标识符类型：int , char , void

len; 标识符数组长度，0表示为变量。

adr; 常量：int为其数值，char为其ascii值，变量为相对偏移量，函数则为本函数内临时变量的个数。

}

extern tab\_entry tab[TABMAX]; 声明该符号表

extern int tab\_idx[TABIDXMAX]; 建立符号表的二级索引

(2) stab字符串常量表

【索引变量】string\_idx

extern std::string string\_tab[SMAX];

文法限定仅当输出语句时，才可以使用字符串。词法分析中读到字符串，就将其登录至stab。用于目标代码生成过程中字符串输出时使用。

## 5．存储分配方案

(1) 运行时的存储组织方案

【概述】

采用动态存储分配的方式。

【具体实现】

为每个程序模块申请一片数据区域（运行栈），并保留该空间直至执行完整个模块为止。

此外应维护一个数据栈。对于局部变量(常量)、形参、全局变量(常量)利用tab表查找找到其在数据栈中的相对当前数据区的基地址偏移；对于临时变量建立临时变量数据表进行存储，根据临时变量的名字查找其所在的数据栈地址偏移。

1. 运行栈的结构

【图示】



【说明】

假设运行栈向上生长。

内容(沿运行栈生长方向)依次是：

全局数据区，用以存放最外层的全局数据（常量、变量）；

ret\_addr存储返回地址，返回地址应为当前指令的下一条指令的地址(pc + 4)；

prev\_abp指向前一个活动记录(AR)的基地址；

存放函数调用的返回值；

存放局部变量和参数；

存放临时变量等信息等。

## 6. 解释执行程序

本难度的编译器不要求实现解释执行程序。

## 7. 四元式设计

(1) 四元式结构体定义

struct midCode{

int op; 操作指令

char z; 结果

char x; 第一个操作数

char y; 第二个操作数

}

(2) 采用的四元式指令表示法

【概述】

四元式指令采用类MIPS汇编指令。具体指令表示方法如下。当然对于部分的跳转指令，或是其他功能的指令，不是严格的4个操作数，这里采用类4元式表示形式。

【表示方式说明】

1） 运算指令

z = x + y 加法指令 add/addi

z = x - y 减法指令 sub

z = x \* y 乘法指令 mul

z = x / y 除法指令 div

z = x 赋值指令 mov

注：赋值指令的三种形式分别为z = y 、 z = y[x] 、 z[x] = y

2） 跳转指令

PC = z 以下设置z为label值，从而指向一个具体代码地址。

beq z, x, y x == y, goto z

bne z, x, y x!=y, goto z

bge z, x, y x>=y, goto z

bgt z, x, y x>y, goto z

ble z, x, y x<=y, goto z

blt z, x, y x<y, goto z

ret y 函数返回 y 值

j z 无条件跳转到label/PC值

jal z 跳转并链接, 紧跟call指令使用，用于处理函数调用

3）其他指令

function x , y 函数声明，y为函数名, x为类型 int, char, void

funcend 函数结束标志

para x , y 函数定义时的形参，x为类型，y为形参名

read y 调用scanf，将值放到y

write x , y 输出字符串x, 表达式的值y

push y 传参y

call y 函数调用操作, y为函数名称

## 8. 目标代码生成方案

(1) 代码生成相关的重要数据结构

- 目标代码存储结构

struct mips\_code{ mips代码项

mipsop op; 指令类型

std::string z; 目标寄存器

std::string x; 操作数寄存器

std::string y; 操作数寄存器

int imm; 立即数

};

extern mips\_code mips\_code\_tab[MIPSCODEMAX]; 目标代码表

(2) 实现代码生成的关键算法

1、操作数处理算法

应用函数：midoptyp midop\_analyse(std::string op , int& offset)

功能：对于z,x,y三个操作数，根据其类型的不同执行不同的操作：

- 若为数字，则直接获取其数值并返回。

- 若为临时变量，则从临时变量区中寻找其位置；在寄存器中存储，则获取寄存器编号，若在数据存储区存储，则使用lw将其加载到一个未使用的寄存器中，并获取此寄存器的编号。

2、字符串输出处理与全局命名规则

应用函数void gen\_mips\_str()

- 为string\_tab表中出现的字符串按顺序编号并命名为str\_n

- 计算出现的字符串占用空间的总大小并向4对齐，用于计算全局变量起始 位置。

3、标签处理方案

首先应决定标签种类，根据不同标签需要实现的功能生成不同标签。

- 循环、分支控制标签：设置标签并决定控制流。

- 函数入口控制：设置$fp(函数基址)和$sp(栈顶)的值，分配全局寄存器。

- 函数起始标签：根据函数名生成标签，命名为func\_name。

- 常量定义标签：.data域中生成一些需要打印输出的字符串标签和数组标签。

4、函数处理方案

- 函数声明处理：在函数声明开始处维护$sp与$fp指针，函数声明内部语句按语句处理方式处理。

- 函数调用处理：使用jal指令跳转并链接。

- 函数返回处理：处理返回值，从栈中恢复调用时刻状态，使用jr $ra指令。

## 9. 优化方案

(1) 常数合并传播

在表达式计算中，将常量替换为其对应的值参与计算，对表达式中的常数计算用其计算结果来代替表达式，从而达到简化运算步骤的效果。

(2) 全局寄存器分配及临时寄存器调度策略

对于每一出现的临时变量采取压栈处理，每次为其分配3个寄存器即可实现功能。

## 10. 出错处理

【处理方案】

对于在编译阶段出现的错误，需要划分为两种不同的情况进行处理。

其一，对于编译中发现的不符合语法语义的错误、非法字符错误等不会导致编译过程崩溃的错误类型，将采取报错并在一定限度内容错的机制。编译过程继续执行，并继续对源代码进行分析。

其二，对于编译中出现的严重错误，比如导致表存储空间不足，或数据溢出的情况。这种“致命”错误，将会导致编译过程崩溃。为了保护程序的安全，需要强制终止编译过程。

【报错信息】

**1.error 类型的错误信息:**

std::string errormsg[70] = {

"缺少类型定义 int 或 char.", // 0

"语句结尾缺少分号", // 1 在变量声明时直接赋值也会报告此错误

"标识符缺失", //2

"缺少赋值符号", //3

"缺少常数数值", //4

"声明类型与变量类型不匹配", //5

"常量定义不完善", //6

"常量定义必须位于变量定义与函数定义之前", //7

"变量定义识别到非法符号", //8

"变量类型不能为空", //9

"数组长度定义必须为整数", //10

"缺少右中括号", //11

"缺少逗号或分号", //12

"变量定义必须位于函数定义之前", //13

"主函数返回值类型必须为void", // 14

"缺少右小括号", //15

"缺少左大括号", //16

"未识别到正确的语句列", //17

"未识别到正确的语句结尾，分号或右大括号缺失", //18

"缺少左小括号", //19

"条件缺失", //20

"未识别到正确的语句头", //21

"未识别到正确的表达式头符号", // 22

"表达式结构错误", //23

"项结构错误", //24

"缺少右中括号", //25

"缺少整数值", //26

"因子结构错误", //27

"写语句内容错误", //28

"返回值表达式错误", //29

"赋值语句缺少赋值符号", // 30

"常量值错误", //31

"缺少冒号", //32

"缺少右大括号", //33

"缺少case关键字", //34

"词法分析读到非法字符", //35

"字符串不以“结尾", //36

"整数以前导零开始", //37

"字符类型出现非法字符", //38

"字符类型不以'结尾", //39

"变量常量重定义错误", //40

"函数未定义", //41

"函数返回值类型错误", // 42

"数组下标不能为字符", //43

"变量未定义", //44

"函数重定义错误", //45

"数组越界", //46

"传入参数过多", //47

"参数类型不匹配", //48

"传入参数过少", //49

"该变量不是数组类型", // 50

"单独出现的数组名错误", //51

"不能为常量赋值", //52

"赋值类型不正确", //53

"条件表达式参数类型错误", //54

"常量类型错误", //55

"不正确的文件结尾", //56

};

**2.fatal:**

std::string fatalmsg[10] = {

"符号表溢出", //0

"中间代码表溢出", //1

"目标代码表溢出", //2

"读取源代码文件异常" //3

};

# 三．操作说明

## 1．运行环境

(1) CodeBlocks 17.12

(2) Mars 4.5

(3) Mars中设置以main为起始位置，取消延迟槽，内存地址格式使用default

## 2．操作步骤

(1) 在CodeBlocks17.12中启动，调整编译器使之适应C++11标准。

(2) 运行origin.cpb，在控制台窗口输入测试文件名。

(3) 目标代码存放在16061160\_mips\_code.txt中。

(4) 在其他输出文件中可以查看中间结果。

(5) 在Mars中运行目标代码，从控制台查看目标代码的执行效果。

(6) 若源程序成功编译，则在控制台显示Compile succeed，否则报告错误信息。

# 四．测试报告

## 1．测试程序及测试结果

【正确性验证测试集】

16061160\_test1.txt

16061160\_test2.txt

16061160\_test3.txt

16061160\_test4.txt

16061160\_test5.txt

【出错处理测试集】

16061160\_error1.txt

16061160\_error2.txt

16061160\_error3.txt

16061160\_error4.txt

16061160\_error5.txt

报错结果将直接显示在控制台。

## 2．测试结果分析

【正确性验证测试集说明】

(1) test1 读操作测试

- 对scanf覆盖测试

- 先连续输入10个数字有回车，再连续输入8个字符无回车

- 预期结果按顺序输出上述内容，前缀2\_100T

- 测试了读入到不同位置的情况（全局，局部，数组前、后）

- 测试了读入类型（int,char)

- 输出类型覆盖了字符串，常量，局部变量，全局变量（int,char全部情况）

- 期望输出与读入内容及顺序一致

测试表明结果正确。

预期输入：

1

2

3

4

5

6

7

8

9

0

abcdefgh

预期输出：

2\_100T

a = 1

b = 2

c = 3

f = 4

g = 5

a1 = 6

b1 = 7

c1 = 8

f1 = 9

g1 = 0

ch1 = a

ch2 = b

ch3 = c

ch5 = d

ch7 = e

ch8 = f

ch9 = g

ch11 = h

(2) test2 赋值，表达式，写操作测试

- 由于printf中含有表达式，故以赋值为载体对表达式也进行测试。

- 用到赋值操作，但只检查简单的表达式。

- 测试期望按照赋值顺序进行输出。

- 测试覆盖了数组的使用情况检查。

- 数组下标只检查了只有数字的情况。

测试表明结果正确。

预期输出：

global\_single\_int =2

global\_int\_1 = 3

global\_int\_2 = 4

global\_single\_char =\_

global\_char\_1 = +

global\_char\_2 = 9

local\_single\_int =100

local\_int\_1 = 101

local\_int\_2 = 102

local\_single\_char =-

local\_char\_1 = \*

local\_char\_2 = /

a = 1

b = 2

c = 3

d[4] = 4

d[5] = 5

e[6] = 6

f = 7

g = 8

e[9] = 9

ch1 = a

ch2 = b

ch3 = c

ch4[1] = A

ch4[2] = B

ch5 = d

ch6[3] = C

ch6[4] = D

a1 = 11

b1 = 12

c1 = 13

d1[4] = 14

d1[5] = 15

e1[16] = 16

f1 = 17

g1 = 18

e1[19] = 19

ch7 = e

ch8 = f

ch9 = g

ch10[1] = E

ch10[2] = F

ch11 = h

ch12[13] = G

ch12[14] = H

(3) test3 表达式综合测试

- 函数调用只检查了最简单的不传参的情况。

- 期望输出为各个表达式的运算结果

测试表明结果正确。

预期输出：

a = 10

b = -11

c = 10

d[0] = 11

d[1] = 11

d[2] = -1

d[3] = 13

f1 = 13

g1 = 22

f = 22

g = 22

e[0] = 3

e[1] = 13

e[2] = 6

e[3] = 22

e1[0] = 98

e1[1] = 130

(4) test4 条件语句测试

- 覆盖条件的每种情况

- 测试了循环语句（条件是同上的）

- 测试了情况语句

测试表明结果正确。

预期输出：

# 1 # 3 # 5 # 7 # 9 # 11 # 13 # 15 1 2 3 4 5 6 7 8 9 10 8 6 4 2 # 17 # 19 # 21 # 23

(5) test5 函数综合测试

- 覆盖函数返回值类型

- 检查多个函数定义及其之间的调用关系

- 检查递归调用

测试表明结果正确。

预期输出：

| print\_num called: 1 x = 4

| print\_num called: 2 x = 5

| print\_num called: 3 x = 6

| print\_num called: 4 x = 7

| print\_num called: 5 x = 8

| print\_num called: 6 x = 9

| print\_num called: 7 x = 10

| print\_num called: 8 x = 1

| print\_num called: 9 x = 2

| print\_num called: 10 x = 3

| print\_num called: 11 x = 4

| print\_num called: 12 x = 26

| print\_num called: 13 x = 8

| print\_num called: 14 x = 21

| print\_num called: 15 x = 9

+

q

y

1

120

【出错处理测试集说明】

出错处理测试集在正确性验证集test5的基础上进行改写，目的是选取一个规模相对比较完善，覆盖语法成分较多的测试用例，在此基础上修改出错误成分。由此构造的测试更具有代表性。

(1) error1 语法错误检查

- 缺少分号检查

- 缺少括号检查（包括缺少左、右小括号、中括号或大括号）

- 程序中出现不正确的语法成分

- 变量名重定义错误

第7行行尾缺少分号错误

第23行数组元素表达缺少中括号

第30行缺少左大括号

第60行缺少左小括号

第123行写语句出现不正确的语法成分

(2) error2 声明错误检查

- 常量定义出现顺序错误

- 变量定义出现顺序错误

- 常量初值缺失

- 为常量赋值错误

第2行常量定义缺少初值

第14行全局常量定义出现顺序错误

第21行全局常量定义出现顺序错误

第26行局部变量定义出现顺序错误

第30行为常量赋值错误

(3) error3 函数错误检查

- 检查函数参数传递是否正确

- 参数个数、类型是否匹配

- 检查函数返回值是否与声明的返回类型相匹配

- 函数调用的函数名是否有声明

第26行缺少函数返回值

第35行函数返回值类型与声明类型不一致

第118行函数调用参数过多

第119行函数调用参数过少

第120行参数类型不正确

第122行函数调用未定义函数名

(4) error4 数组错误检查

- 下标整数越界检查

- 下标常量越界检查

- 表达式左部越界检查

- 表达式右部越界检查

- 数组下标不为整数检查

第24行表达式右部数组下标整数越界

第25行表达式右部数组下标整数越界

第26行表达式右部数组下标常量越界

第27行表达式右部数组下标为字符型错误

第28行变量定义顺序错误

第38行表达式左部数组下标整数越界

第39行表达式左部数组下标整数越界

第40行表达式左部数组下标常量越界

第41行表达式左部数组下标为字符型错误

第42行变量定义顺序错误

(5) error5 类型错误检查

- 条件表达式中出现char类型

- 赋值等式两端数据类型不一致

- 表达式两端比较类型不一致

- 函数返回值类型与声明类型不一致

第27行字符常量为整型变量赋值错误

第29行字符值为整型变量赋值错误

第35行条件表达式中出现字符错误

第71行条件表达式中出现字符常量错误

# 五．总结感想

计算机学院开设的每一个课程设计，都会让我收获满满，编译技术课设，从其设计要求，到具体的实现过程，再到后来的考核与答疑，都让我十分满足，这门课程对得起“国家级精品课程”的荣誉！

以下来谈谈我完整的做完了整个编译器以后，我的体会以及收获。

最大的收获当然是编程能力的提升。在大一的时候学习数据结构和算法时，我们曾初涉C语言，但是了解并不深入。在本学期我们的课程设计要求使用C语言从头开始编写一个完整的编译器，这对我来说是个不小的挑战。工程规模大、实现复杂、调试难度大，都是我在编写的过程中遇到的困难。在这个过程中，我开始接触并渐渐熟悉了C++的编程思路，了解到了C++的一些很便捷的特性，使代码更加简洁已读；我更进一步学习了头文件的编写方法，使整个工程可以分为若干个文件，每个文件只关注其负责的特定的业务逻辑，使自己的代码更加符合工程规范。

另外我完整的做完了我们的实验以后，对理论课所学内容也有了更近一步的认识。按照老师的说法，“一学期三次学习编译原理”的过程，让我在每一轮学习的过程中都有着新的体会。我也从中深刻的体会到了编译技术课程改革的重大优势。首先学完基础篇的内容之后，我就可以完成一个编译器的编写了，原来在课本上的“编译的7个逻辑部分”也通过实验的编写而“活”了起来。随后又利用了提高篇中提到的优化算法对编译器的中间代码进行了优化，训练了自己的算法落地能力。

最后不得不提到的就是我从中训练出的良好的心态和时间规划能力。编译技术课程设计的时间节点要求十分严格，因此我必须提前规划好自己的时间。同时，编译实验也伴随着许多其他大作业和考试的开展同时进行，我必须平衡好各个工作投入的精力。即使在压力最大的时候，我也必须努力去调整自己的心态，学会坚持，学会迎接挑战！

最后谈谈对课程设计的建议。我觉得整个实验的安排相对而言是比较合理的，只有一点在我真正实现过程中让我有些困扰。关于论坛的使用，我希望老师能在下一届的教学当中进行规范，论坛更多的应该是增加同学时间的技术交流，多多关注涉及编译技术本身的问题加以探讨，而我不幸地注意到，有部分同学在论坛上提问了大量与技术本身无关，而只深挖实现细节的问题。其实对于一项工程来讲，实现功能才是最为重要的目的，正确性是由工程师自己设计保证的，我希望作为北航的毕业生，我们能够多一些自己独立的思考，关于什么是“正确”，应该有一套自己的理解，并正确地去实现，而绝非不经深思熟虑，即上论坛发帖求助，用别人的大脑来代替自己的思考。对于这一点，我希望今后能有有效的措施来规法大家对于论坛的使用。

但总体而言，编译技术课程设计是我在北航计算机学院所学到的最满意的一门课程之一，我衷心的希望这门课程能越来越好，使更多学弟学妹们能从中受益！

**陈麒先**

2019/1/4