《编译技术》课程设计文 档

14061124

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

杨佳琦

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

年 月 日

## 一．需求说明

### 1．文法说明

＜加法运算符＞ ::= +｜-

定义加减法运算符

限定条件：只有+和-

例子：3+4 a+b 3+a

＜乘法运算符＞ ::= \*｜/

定义乘除法运算符

限定条件：只有\*和/

例子：1/2 a/2 a/b

＜关系运算符＞ ::= <｜<=｜>｜>=｜!=｜==

定义关系运算符

限定条件：只有这几种

例子：1<2 a>=3 b!=8

＜字母＞ ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

定义字母可以为大小写字母或者下划线

限定条件：除了大小写字母外，还包含了下划线\_ ，而其他特殊符号不包含

例子：\_ a B

＜数字＞ ::= ０｜＜非零数字＞

定义数字为0~9的单位数字字符

限定条件：0或非零数字

例子： 0 1 8

＜非零数字＞ ::= １｜．．．｜９

定义非零数字为1~9的单位数字字符

限定条件：1~9

例子： 1 8

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

定义字符为以单引号开始和结束，其间有且仅有一个字符。

限定条件：

1. 字符必须是以单引号'作为首尾端。

2. 中间的部分是之前声明的那些符号。

3. 中间只能有一个符号，总长度是3。

例子： '+' '/' 'a' '8'

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

定义字符串可以为0个，1个，或多个十进制编码为32,33,35-126的ASCII字符

需要注意的是不包括双引号,但是包括空格

字符串的第一个和最后一个字符为双引号

例子："hello, world~"

＜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＞

一个程序的组成部分

限定程序的子成分顺序，必须为常量说明在前，然后是变量说明，以上两者均可没有。

然后是有返回或者无返回的函数定义若干（可以是0个），然后才是主函数

例子: const int a = 3;

int b = 4;

void f(int a);

int f2();

void main(){...}

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

定义常量说明可以是一个或者多个const<常量定义>；这样的格式

1. 一个const后面只能跟着一种类型变量的定义。

2. const间以分号;隔开。

3. 末尾一定是有分号的。

例子： const int a = 2;

const char c = 'b';

＜常量定义＞ ::= int＜标识符＞＝＜整数＞{,＜标识符＞＝＜整数＞}

| float＜标识符＞＝＜实数＞{,＜标识符＞＝＜实数＞}

| char＜标识符＞＝＜字符＞{,＜标识符＞＝＜字符＞}

定义常量定义可以是整型常量或者字符型常量

限定条件：

1.每次只能为一种类型变量进行定义。

2. int/char/float 后面的赋值语句最少得有一个。

3. 不允许只定义不赋值情况的出现： int i

整型常量的定义格式是int后跟一个或多个＜标识符＞＝＜整数＞的形式，之间用逗号隔开

字符型常量同理

例子： int a = 2;

Float b= b;

char c = 'c';

＜无符号整数＞ ::= ＜非零数字＞｛＜数字＞｝

定义无符号整数的开头必为非零数字，其后可跟若干个数字

该限定表明以0开头的数字串不属于无符号整数

0就不是无符号整数

例子： 1 99

＜整数＞ ::= ［＋｜－］＜无符号整数＞｜０

定义整数是以可省略的正负号开头，后跟无符号整数或者0的字符串

限定条件：

1. 0前面不能有+-

例子： 0 1 3

＜小数部分＞ ::= ＜数字＞｛＜数字＞｝｜＜空＞

定义小数的开头必为非零数字，其后可跟若干个数字，可为空

例子：0.1 2.33

＜实数＞ ::= ［＋｜－］＜整数＞[.＜小数部分＞]

定义整数是以可省略的正负号开头，后面跟整数和小数部分

例子：+1.2 -2.2 +1

＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝

定义标志符是必须由字母为开头，后跟0到多个字母或者数字的字符串

限定条件：

1. 由字母和单词组成。

2. 首位一定是字母。

示例：

a

\_

a2\_A0

\_2a\_

＜声明头部＞ ::= int＜标识符＞ |float ＜标识符＞|char＜标识符＞

定义声明头部

限定条件：

1. 只有int和char两种类型。

示例：

int a

char abc

＜变量说明＞ ::= ＜变量定义＞;{＜变量定义＞;}

定义变量说明为一个或多个变量定义和分号组成的字符串

限定条件：

1. 每条变量定义语句后都要有一个分号;

示例：

int i1,i2,is[123];

char c1,c2[12],c3;

int i1; char c1;

＜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＞‘]’ )}

定义变量定义，变量可以为一个标志符或者由标志符为起始的数组形式。

限定条件：

1. 一个变量定义语句只能指定一种类型的变量，变量可以是数组，变量间以逗号,分隔。

2. ＜类型标识符＞后面至少有一个变量定义。

例子： int a2bc\_d

int word[10]

但是不能如下定义：

int 34dfn

int word[-1]

＜可枚举常量＞ ::= ＜整数＞| ＜字符＞

作用：指明常量的组成。

限定条件：

1. 只能是<整数>或<字符>

示例：

0

+123

-123

'a'

'\_'

＜类型标识符＞ ::= int | float | char

定义类型标识符为整数和字符

限定条件：

1. 只有int和char两种。

示例：

int

char

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

定义有返回值函数。必须包含头声明头部，参数和复合语句以及必要的括号

限定条件：

1. 组成中的三部分都必须要有，而且顺序不能改变。

2. 即使＜参数＞和＜复合语句＞为<空>，()和{}也要有。

例子： int f(char a){

...

...

}

char f(char a){

...

...

}

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

定义无返回值函数

限定条件：

1. 必须是以void开头，组成部分的顺序不能改变。

2. 即使＜参数＞和＜复合语句＞为<空>，()和{}也要有。

例子： void f(char a){

...

...

}

＜复合语句＞ ::= ［＜常量说明＞］［＜变量说明＞］＜语句列＞

作用：指明复合语句的组成和格式。

限定条件：

1. 如果＜常量说明＞和＜变量说明＞都存在的话，他们的顺序不能变。

示例：

//空

;

s=1;

s=1;if(id1==id2) s=0;while(id1<id2) s=s+1;

const int i = 1;

const int i = 1;;

const int i = 1;s=1;

int i1,i2[13];

int i1,i2[13];;

int i1,i2[13];s=1;

const int i = 1; int i1,i2[13];

const int i = 1; int i1,i2[13];;

const int i = 1; int i1,i2[13];s=1;

＜参数＞ ::= ＜参数表＞

作用：声明参数。

限定条件：无

示例：

//空

int i

int i, char c

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

参数表由若干个类型标识符和标志符的集合组成，其间用逗号隔开。可以为空。

类型标识符和标志符的顺序不能反。

需要注意的事不能有数组形式

例如： int a, char b, int c,

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

作用：指明一个主函数的组成。

限定条件：

1. 除了<复合语句>之外，其他都是固定的内容。

例子： void main() {

...

}

＜表达式＞ ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞}

表达式是由可省略的+-开头的若干个由加法运算符连接的项的字符串

限定条件：

1. <项>之间必须有 + or -

2. 至少要有一个<项>

3. 第一个<项>之前可以有+ or -

例如： 3\*4+2/6-4\*a

＜项＞ ::= ＜因子＞{＜乘法运算符＞＜因子＞}

项是由乘法运算符连接的一个或多个因子

作用：指明一个项的组成与结构。

限定条件：

1. <因子>间必须要有\* or /

例如： 4\*b\*a

13\*6\*num[15]

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

因子是由标志符或标志符后跟方括号括起来的表达式或整数或者字符或者有返回值得函数调用语句或以圆括号括起来的表达式

限定条件：

1. 不能为空

例如： num[15]

num[a]

a

4

(3\*13+5\*a-2/f)

＜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＞;

作用：指明语句的组成。

限定条件：

1. 除了‘{’＜语句列＞‘}’ 和 <情况语句>外，最后一个字符一定是;

if(id1==id2) s=0; else s=1;

while(id1<id2) s=s+1;

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

作用：指明赋值语句的组成。

限定条件：非空

例如： a[3] = 4

b = 5

＜条件语句＞ ::= if ‘(’＜条件＞‘)’＜语句＞［else＜语句＞］

作用：指明条件语句的组成。

限定条件：

1. 条件不能为空。

2. else部分可有可无，不支持 else if语句。

例如： if(a > b){

a = 1;

b = 2;

}

＜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

表达式为0条件为假，否则为真，值得注意的是一个条件中只有一个关系运算符，不支持嵌套

例如：

1&2 a&b a 0 1

＜循环语句＞ ::= while ‘(’＜条件＞‘)’＜语句＞

作用：指明循环语句的组成。

循环语句具有如下形式：

while(...){

...

}

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

作用：指明情况语句的组成。

限定条件：

1. 缺省部分可有可无。

示例：switch (id){case 1:s=1;case 2:s=2;default:s=0}

＜情况表＞ ::= ＜情况子语句＞{＜情况子语句＞}

作用：指明情况表的组成。

限定条件：

1. 至少得有一个情况子语句

示例：

case 1:s=1;

＜情况子语句＞ ::= case＜可枚举常量＞：＜语句＞

作用：指明情况子语句的组成。

限定条件：

1. case后面只能跟int和char

2. 不需要break,执行完当前子语句不会再往下执行。

示例：

case 1: s=1;

＜有返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’

作用：指明有返回值函数调用语句的组成

例如： f(2, 3)

＜无返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’

作用：指明无返回值函数调用语句的组成

例如： f(2, 3)

＜值参数表＞ ::= ＜表达式＞{,＜表达式＞}｜＜空＞

值参数表是若干个（包括0）表达式的集合

例子： //空

1

'a'

id

id1,id2,32,'a',fun()

＜语句列＞ ::= ｛＜语句＞｝

语句列是若干（包括0）个连续语句的集合

例子：s=1;

s=1;if(id1==id2) s=0;while(id1<id2) s=s+1;

＜读语句＞ ::= scanf ‘(’＜标识符＞{,＜标识符＞}‘)’

定义读语句是以scanf起始，后接圆括号阔起来的一个或多个以逗号隔开的标识符。

例子: scanf(a, b)

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

定义写语句是以printf为起始的，后接圆括号括起来的字符串或表达式或者两者都有，若两者都存在，则字符串在先，以逗号隔开。

例子： printf("%d", a)

printf(a)

printf("Hello, compilier!\n")

＜返回语句＞ ::= return[‘(’＜表达式＞‘)’]

返回语句是以return开始，后接可有可无的以圆括号包围的表达式

例子： return 0

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）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

2．目标代码说明

Pcode码。

 P-code 语言：一种栈式机的语言。此类栈式机没有累加器和通用寄存器，有一个栈式存储器，有四个控制寄存器（指令寄存器 I，指令地址寄存器 P，栈顶寄存器 T和基址寄存器 B），算术逻辑运算都在栈顶进行。

|  |  |  |
| --- | --- | --- |
| F | L | A |

        指令格式

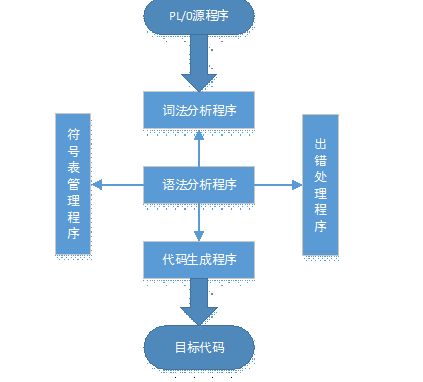
       F ：操作码

       L ：层次差（标识符引用层减去定义层）

       A ：不同的指令含义不同

## 二．详细设计

### 程序结构



### 2．类/方法/函数功能

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | pl0 | 主程序 | | error | 出错处理，打印出错位置和错误代码 | | getsym | 词法分析，读取一个单词 | | getch | 取字符 | | gen&genf | 生成P-code指令，送入目标程序区 | | enterValue | 标识符类型 | | enterStart | 代码开始 | | enter | 登记符号表 | | cleanTable | 清空符号表 | | find | 查是否有重名 | | Lookup | 查找在符号表中的位置 | | findGlobal | 找全局变量 | | statement | 语句部分分析处理 | | expression | 表达式分析处理 | | terms | 项分析处理 | | factor | 因子分析处理 | | Ifstatement | ifelse分析处理 | | Whilestatement | While分析处理 | | Callorassign | 调用或赋值分析处理 | |
| |  |  | | --- | --- | | Readstatement | 读分析处理 | | Writestatement | 写分析处理 | | Switchstatement | Switch分析处理 | | Returnstatement | 返回分析处理 | | Valueofpara | 参数个数 | | Vardefine | 变量说明 | | vardefineMain | Main定义 | | Constdefine | 常量说明 | | Mainfun | Main函数执行 | | Program | 程序调用 | | clearToken | 清空token | | Transnum | 转化成数 | | isLetter | Letter判断 | | isDigit | 是否是数字 | | Integerjudge | 整形判断 | | Realnumjudge | 实数判断 | | Interupt | P-code解释执行程序 | | Main | 主函数 | |

### 3调用依赖关系

1. pl0
2. getsym
3. getch
4. Gen
5. genf
6. Integerjudge
7. realnumjudge
8. Table
9. cleantable
10. Enter
11. enterValue
12. enterStart
13. cleanTable
14. Readstatement
15. Ifstatement
16. Whilestatement
17. Switchment
18. Vardefine
19. comstdefine
20. Vardefinemain
21. Callorassign
22. Returnstatement
23. Writestatement
24. Mainfun
25. program
26. statement
27. expression
28. term
29. factor
30. Find
31. Lookup
32. findGlobal
33. error
34. interpret
35. main

### 4．符号表管理方案

### 符号表数据结构如下：

符号表中每一条记录所对应的结构：

### struct tablestruct

### {

### char name[20]; /\*名字\*/

### int level; /\*所处层\*/是指变量为全局变量还是局部变量

### int type; /\*标识符\*/类型 void int float char

### int size; /\*数组长度\*/

### int kind; /\*类型：int float char fun\*/常量还是变量

### union Valuetype valuetype;

### int offset; //地址偏移

### int ret; //是否是数组

### int parameter; /\*参数个数\*/

相关函数：void enter(int add,char \*name,int level,int type,int kind,int size,int ret){//添加到符号表

void enterValue(int type){//标识符的类型

int find(char\* name)查找是否重名

int lookup(char \*name)查找符号表中的位置

### };5．存储分配方案

类PCODE代码解释执行的部分通过循环和简单的case判断不同的指令，做出相应的动作。当遇到主程序中的返回指令时，指令指针会指到0位置，把这样一个条件作为终至循环的条件，保证程序运行可以正常的结束。

### 运行栈的存储分配

### .SL:静态链，指向定义该过程的直接外过程(或主程序)运行时最新数据段的基地址。

### .DL:动态链，指向调用该过程前正在运行过程的数据段基地址。

### .RA:返回地址，记录调用该过程时目标程序的断点，即调用过程指令的下一条指令的地址

### 解释执行类PCODE代码时，数据段存储分配方式如下：

### 对于源程序的每一个过程（包括主程序），在被调用时，首先在数据段中开辟三个空间，存放静态链SL、动态链DL和返回地址RA。静态链记录了定义该过程的直接外过程（或主程序）运行时最新数据段的基地址。动态链记录调用该过程前正在运行的过程的数据段基址。返回地址记录了调用该过程时程序运行的断点位置。对于主程序来说，SL、DL和RA的值均置为0。静态链的功能是在一个子过程要引用它的直接或间接父过程（这里的父过程是按定义过程时的嵌套情况来定的，而不是按执行时的调用顺序定的）的变量时，可以通过静态链，跳过个数为层差的数据段，找到包含要引用的变量所在的数据段基址，然后通过偏移地址访问它。

### 在过程返回时，解释程序通过返回地址恢复指令指针的值到调用前的地址，通过当前段基址恢复数据段分配指针，通过动态链恢复局部段基址指针。实现子过程的返回。对于主程序来说，解释程序会遇到返回地址为0的情况，这时就认为程序运行结束。

### 

### 6. 解释执行程序\*

### 类pcode解释器的结构

. 目标代码存放在数组code[]中

. enum fct p;//指令寄存器

pc;//指令地址寄存器

mbase=0;//基地址寄存器

mtop=0;//存储栈栈顶寄存器

struct arun{

int isfloat;

int zheng;

double shi;

}run[500],mem[500]; //运行栈

int rtop=0;//运行栈栈顶寄存器

P-code指令：

LIT 0 A,将立即数A存入栈顶

ADD 0 0,栈顶与次栈顶相加，结果存于次栈顶

SUB 0 0,次栈顶减栈顶，结果存于次栈顶

MUL 0 0,次栈顶和栈顶乘，结果存于次栈顶

DIV 0 0,次栈顶除以栈顶，结果存于次栈顶

LOD l A,将层次为l，相对地址为A的变量置于栈顶

LODT l 0,将地址为栈顶的变量取出置于栈顶,层次为l

LOA l A,将层次为l,相对地址为A的地址取出置于栈顶

STT l 0,将栈顶数存于次栈顶地址中,层次为l

LOG 0 A,逻辑运算集合，进行次栈顶与栈顶的逻辑运算 结果存于次栈顶

顺序依次为1 > 2 >= 3 < 4 <= 5 != 6 ==

LOGT 0 A逻辑运算集合，进行次栈顶与栈顶的逻辑运算 结果存于栈顶

顺序依次为1 > 2 >= 3 < 4 <= 5 != 6 ==

JPC 0 A,若栈顶为0 则跳转到指令A

JMP 0 A,无条件跳转到A

INT 0 A,存储栈栈顶开辟A个存储单元

STI l A,将立即数l存于相对地址A中

JMB 0 0,跳至最近的ret addr处

STO l A,将栈顶元素存于层次为l，相对地址为A的存储区

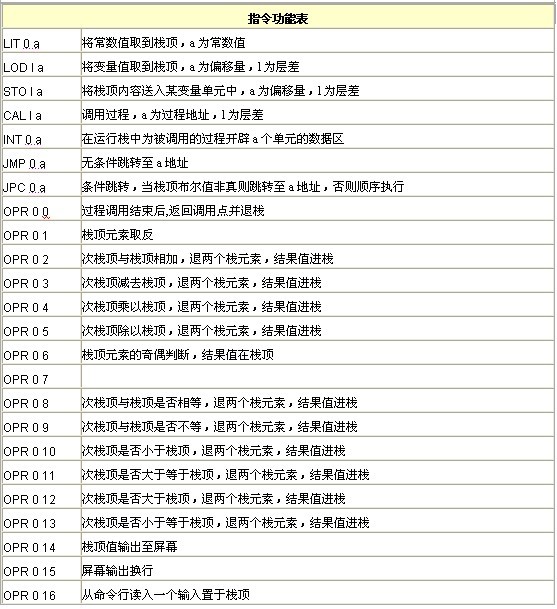
QUT 0 0,存储栈退栈一次

rei 0 0

ref 0 0

rec 0 0读取指令，从标准输入输出读取,存于栈顶

wr 0 0标准输出指令，输出栈顶至标准输出



### 8. 目标代码生成方案\*

 P-code ：栈式机没有累加器和通用寄存器，有一个栈式存储器，有四个控制寄存器（指令寄存器 I，指令地址寄存器 P，栈顶寄存器 T和基址寄存器 B），算术逻辑运算都在栈顶进行。

|  |  |  |
| --- | --- | --- |
| F | L | A |

        指令格式

       F ：操作码

       L ：层次差（标识符引用层减去定义层）

       A ：不同的指令含义不同

//pcode类的结构

struct instruction

{

enum fct f;

union pkind l;

union pkind a;

int isfloat;

};

//虚拟机代码指令

//引用层与声明层的层次差

//指令参数

//是否是float

//存放虚拟机代码的数组

struct instruction code[cxmax];

//生成虚拟机代码

void gen(enum fct x,int y,int z){

code[cx].f=x;

code[cx].l.intkind=y;

code[cx].a.intkind=z;

code[cx].isfloat=0;

}

void genf(enum fct x,int y,float z){

code[cx].f=x;

code[cx].l.intkind=y;

code[cx].a.floatkind=z;

code[cx].isfloat=1;

}

### 10. 出错处理

尽可能准确指出错误位置和错误属性和尽可能进行校正

定义一个is\_error，作为标识，无错误时是0，出现错误则改为1.

void error(int i){//

switch(i){

case 1:

printf("error%d,in line%d\n",i,line);

puts("没有=");

break;

case 2:

printf("error%d,in line%d\n",i,line);

puts("少引号");

break;

case 3:

printf("error%d,in line%d\n",i,line);

puts("标识符太长");

break;

case 4:

printf("error%d,in line%d\n",i,line);

puts("识别错误");

break;

case 5:

printf("error%d,in line%d\n",i,line);

puts("不是符合要求的数字");

break;

case 6:

printf("error%d,in line%d\n",i,line);

puts("少类型标识符");

break;

case 7:

printf("error%d,in line%d\n",i,line);

puts("重复定义");

break;

case 8:

printf("error%d,in line%d\n",i,line);

puts("少标识符");

break;

case 9:

printf("error%d,in line%d\n",i,line);

puts("应该是等号");

break;

case 10:

printf("error%d,in line%d\n",i,line);

puts("常量定义类型不匹配");

break;

case 11:

printf("error%d,in line%d\n",i,line);

puts("符号太多了0.0");

break;

case 12:

printf("error%d,in line%d\n",i,line-1);

puts("没有；");

break;

case 13:

printf("error%d,in line%d\n",i,line);

puts("少]");

break;

case 14:

printf("error%d,in line%d\n",i,line);

puts("int");

break;

case 15:

printf("error%d,in line%d\n",i,line);

puts("声明变量错误");

break;

case 16:

printf("error%d,in line%d\n",i,line);

puts("应为类型标识符");

break;

case 17:

printf("error%d,in line%d\n",i,line);

puts("少）");

break;

case 18:

printf("error%d,in line%d\n",i,line);

puts("少{");

break;

case 19:

printf("error%d,in line%d\n",i,line);

puts("少}");

break;

case 20:

printf("error%d,in line%d\n",i,line);

puts("识别错误");

break;

case 21:

printf("error%d,in line%d\n",i,line);

puts("少（");

break;

case 22:

printf("error%d,in line%d\n",i,line);

puts("=错误");

break;

case 23:

printf("error%d,in line%d\n",i,line);

puts("应为变量");

break;

case 24:

printf("error%d,in line%d\n",i,line);

puts("缺少：");

break;

case 25:

printf("error%d,in line%d\n",i,line);

puts("应为void");

break;

case 26:

printf("error%d,in line%d\n",i,line);

puts("应为main");

break;

case 27:

printf("error%d,in line%d\n",i,line);

puts("重复定义");

break;

case 28:

printf("error%d,in line%d\n",i,line);

puts("没有此标识符");

break;

case 29:

printf("error%d,in line%d\n",i,line);

puts("数组越界");

break;

case 30:

printf("error%d,in line%d\n",i,line);

puts("数组错误");

break;

case 31:

printf("error%d,in line%d\n",i,line);

puts("没有找到次函数");

break;

case 32:

printf("error%d,in line%d\n",i,line);

puts("参数类型不对");

break;

case 33:

printf("error%d,in line%d\n",i,line);

puts("不能对其赋值");

break;

case 34:

printf("error%d,in line%d\n",i,line);

puts("switch错误");

break;

case 35:

printf("error%d,in line%d\n",i,line);

puts("赋值错误");

break;

case 36:

printf("error%d,in line%d\n",i,line);

puts("返回类型不对");

break;

case 37:

printf("error%d,in line%d\n",i,line);

puts("参数错误");

break;

case 38:

printf("error%d,in line%d\n",i,line);

puts("因子错误");

break;

case 39:

printf("error%d,in line%d\n",i,line);

puts("返回值与函数类型不符");

break;

case 40:

printf("error%d,in line%d\n",i,line);

puts("类型错误");

break;

case 41:

printf("error%d,in line%d\n",i,line);

puts("weizhi cuowu");

break;

case 42:

printf("error%d,in line%d\n",i,line);

puts("switch 错误");

break;

case 43:

printf("error%d,in line%d\n",i,line);

puts("shao");

break;

}

is\_error=1;

}

## 三．操作说明

### 1．运行环境

Windows系统和软件Codeblocks13.12

### 2．操作步骤

打开工程文件，编译并运行。

在控制台直接输入测试程序路径，回车即可

## 四．测试报告

### 1．测试程序及测试结果

测试程序1

int add(int a,int b){

int c;

c = a+b;

return (c);

}

float adds(float a,float b){

float c;

c =a+b;

return (c);

}

void main(){

int a;

int b;

int c;

int d;

int i;

int x;

int y;

float m;

float n;

scanf(d);

scanf(x);

scanf(y);

scanf(m);

scanf(n);

a=1;

i=0;

b=a+1;

c=a\*b;

switch(d){

case 1:printf(a);

case 2:printf(b);

case 3:printf(x\*y);

}

while(i<5){

c=c+1;

i=i+1;

}

if(x<y){

printf(x);

}

else{

printf(y);

}

printf(c);

printf(add(x,y));

printf(adds(m,n));

}

输入：3 1 2 1 1

输出：2 1 7 3 2.000000

错误1：

int add(int a,int b){

int c

c = a+b;

return (c);

}

float adds(float a,float b){

float c;

c =a+b;

return (c);

}

void main(){

int a;

int b;

int c;

int d;

int i;

int x;

int y;

float m;

float n;

scanf(d);

scanf(x);

scanf(y);

scanf(m);

scanf(n);

a=1;

i=0;

b=a+1;

c=a\*b;

switch(d){

case 1:printf(a);

case 2:printf(b);

case 3:printf(x\*y);

}

while(i<5){

c=c+1;

i=i+1;

}

if(x<y){

printf(x);

}

else{

printf(y);

}

printf(c);

printf(add(x,y));

printf(adds(m,n));

}

输出：error12,in line2

没有；

测试程序2：

void main(){

int a[100];

int i;

int n;

i = 0;

a[0] = 1;

a[1] = 1;

scanf(n);

while(i<n){

if(i>=2){

a[i]=a[i-1]+a[i-2];

}

printf(a[i]);

i=i+1;

}

}

输入：20

输出：feibonaqi shi 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765

错误：

void test(){

printf("feibonaqi shi");

}

void main(){

int a[100];

int i;

int n;

i = 0;

a[0] = 1;

a[1] = 1;

scanf(test);

test();

while(i<n){

if(i>=2){

a[i]=a[i-1]+a[i-2];

}

printf(a[i]);

i=i+1;

}

}

输出：error33,in line11

不能对其赋值

测试程序3：void main(){

int a;

int b;

int i,j;

int n;

scanf(n);

scanf(a);

scanf(b);

j=1;

i=1;

while(i<n){

while(j<n){

if(a<b){

printf(a+b);

}

else{

printf(a/b);

}

j=j+1;

}

i=i+1;

}

}

输入：5 2 3

输出：5555

错误：

void main(){

int a;

int b;

int i,j;

int n;

scanf(n);

scanf(a);

scanf(b);

j=1;

i=1;

while(i<n){

while(j<n){

if(a<x){

printf(a+b);

}

else{

printf(a/b);

}

j=j+1;

}

i=i+1;

}

}

输出：error28,in line14

没有此标识符

测试程序4：

void main(){

int a;

scanf(a);

while(a<=10){

switch(a){

case 1:printf("a is 1");

case 2:printf("a is 2");

case 3:printf("a is 3");

case 4:printf("a is 4");

case 5:printf("a is 5");

case 6:printf("a is 6");

case 7:printf("a is 7");

case 8:printf("a is 8");

case 9:printf("a is 8");

case 10:printf("a is 10");

}

scanf(a);

}

}

输入输出：1

a is 1

2

a is 2

3

a is 3

5

a is 5

9

a is 8

错误：

int g(int aa)

{

int a;

if(aa==1)

a=1;

else

a=aa\*g(aa-1);

}

void main()

{

int i;

int a;

scanf(i);

a=g(i);

printf a;

}

error21,in line15

少（

error17,in line15

少）

测试程序5：

int change[10];

void main(){

change[0+1]=1;

change[1\*2]=2;

change[2+3]=change[1]+change[2];

if(change[1]<change[2])

printf(change[5]);

else

printf("false");

}

输出：3

错误：int change[10];

void main(){

change[0+1]=1;

change[1\*2]="s";

change[2+3]=change[1]+change[2];

if(change[1]<change[2])

printf(change[5]);

else

printf("false");

}

error38,in line4

因子错误

error12,in line3

没有；

### 2．测试结果分析

测试程序一有有返回值函数定义，+-\*运算。读入语句，switch case， while，和if else语句，写语句，是一个比较全面的测试程序。

错误部分就是删掉一个；

测试程序二是个斐波那契数列的计算，主要是检查递归。

错误程序是读语句中读入一个函数名。

测试程序三是两个while循环嵌套加if语句的计算，检查多个语句嵌套的正确性。

错误程序是使用了一个未定义的字符。

测试程序四是switch case和while循环输入

错误程序是少一对括号

测试程序五是对数组进行计算和条件判断。

错误程序是在int型数组中存入一个s。

## 五．总结感想

这次编译作业还是有些遗憾的，由于身体原因没能选择高难度挑战一下自己。虽然是中难度，感觉这个编译器的编写有很多要考虑和注意的地方。比如我经常忘记考虑是否是float类型的数。也有自己考虑不全的地方，测试程序如果最后一行是空行我的程序就会报错。还有就是文法中如果遇到错误的处理，这个和同学交流后能处理43类错误，其实还有一些想到的错误，我自己没有写好就先忽略了，时间紧促，总觉得写的有很多bug。有时间还想再改进一下。再有就是pcode指令的翻译，这部分感觉还是比较麻烦的，我是从网上查了一下有类似的文章讲解，根据查的资料改写的。文法部分逻辑性还是很强的，比较容易构思。写出程序也更好的理解了课上的内容。总之，这个程序很好的锻炼了我，提升了编程的能力。