《编译技术》课程设计文档

学号： 14061103

姓名： 卫爱民

2016年 11 月 10 日

1. **需求说明**
2. **文法说明**

原文法：

扩充C0文法-中-数组-有for没有switch，case

＜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＞}

| float＜标识符＞＝＜实数＞{,＜标识符＞＝＜实数＞}

| 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

＜小数部分＞ ::= ＜数字＞｛＜数字＞｝｜＜空＞

＜实数＞ ::= ［＋｜－］＜整数＞[.＜小数部分＞]

＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝

＜声明头部＞ ::= int＜标识符＞ | float ＜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＞‘]’ )}

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

＜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＞;

＜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＞［else＜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＞| for‘(’＜标识符＞＝＜表达式＞;＜条件＞;＜标识符＞＝＜标识符＞(+|-)＜步长＞‘)’＜语句＞

＜步长＞ ::= ＜非零数字＞｛＜数字＞｝

＜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＞‘)’]

文法分析：

1、＜加法运算符＞ ::= +｜-

分析：加减法运算符定义，可以是+或者-

样例：+ -

2、＜乘法运算符＞  ::= \*｜/

分析：乘法运算符定义，可以是\*或者/

样例：\* /

3、＜关系运算符＞  ::=  <｜<=｜>｜>=｜!=｜==  
分析：关系运算符定义，可以是<,<=,>,>=,!=,==号

样例：a > b, A != B

4、＜字母＞   ::= ＿｜a｜．．．｜z｜A｜．．．｜Z  
分析：字母的定义，可以是a到z，A到Z中的任意一个字母

样例：a ， A

5、＜数字＞   ::= ０｜＜非零数字＞

分析：数字的定义，数字分为两类，0或者非零数字

样例：0 ，1， 2

6、＜非零数字＞  ::= １｜．．．｜９

分析：进一步定义非零数字，非零数字就是1~9这9个数

样例：1, 5， 7

7、＜字符＞    ::=  '＜加法运算符＞'｜'＜乘法运算符＞'｜'＜字母＞'｜'＜数字＞'  
分析：字符是包括四种，加法运算符，乘法运算符，字母以及数字

样例：’+’ ，’2’ ，’a’ ，’/’

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

分析：字符串是空格，感叹号，井号等出现若干次的字符序列，大括号表示出现任意次，所以字符串可以为空

样例：”” ,”12a” ,”#dda$!@”

9、＜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：程序定义，程序是由常量说明，变量说明以及有返回值函数的定义，无返回值函数的定义以及主函数组成，其中常量说明和变量说明可有可无，函数的定义可以出现任意次。

样例：

const float PI = 3.14;

int a = 0;

int count()

{

程序体

}

Void add()

{

程序体

}

Int main()

{

主函数体

}

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

分析：常量说明是由const保留字为首的一系列常量定义，这个结构可以出现大于等于一次。

样例： const int a=0;

Const float b=0.0;

11、＜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＞}

分析：常量定义包括几种类型的数据定义，包括int，float，char类型的数据定义，定义的格式是当前数据类型的保留字加上所要定义的标识符，最后=加上给予的初值。其中,＜标识符＞＝＜整数＞的结构可以出现大于等于一次，表示所有后面定义的标识符都是这个类型。

样例：int a = 0,b =1

Float c = 0.0,d = 0.01

Char h = ‘a’

12、＜无符号整数＞  ::= ＜非零数字＞｛＜数字＞｝  
分析：无符号整数是由非零数字的后面加上若干位的数字组成的，这里的无符号整数不能为0。

样例： 100 ， 123 ，9999999

13、＜整数＞        ::= ［＋｜－］＜无符号整数＞｜０

分析：整数是由前面带有正号或者负号或者没有符号的无符号整数或者0构成，这里0前面不能加正号或者负号。

样例： +999 ， -213, 0

14、＜小数部分＞    ::= ＜数字＞｛＜数字＞｝｜＜空＞  
分析：小数部分是由一位数字后面加上若干位数字或者空组成。

样例：12313123 ， ， 1212 ，000

15、＜实数＞        ::= ［＋｜－］＜整数＞[.＜小数部分＞]

分析：实数是由前面带有正号或者负号或者前面没有符号的整数，以及后面带上或者不带上逗号加小数部分组成的，这里负号可以出现两次，因为整数前面可以有符号实数前面也带有符号，0前面只能跟一个或者没有符号。

样例： +121.00 , -0.1, +-1231, - 12.00 , 6.

16、＜标识符＞    ::=  ＜字母＞｛＜字母＞｜＜数字＞｝

分析：表示符是由字母打头的，后面可以跟上若干个数字或者字母的串组成。

样例：a1,while,intd12

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

分析：声明头部是由数据类型（int，float，char）其中一种以及标识符组成。

样例： int a1 , float ab

18、＜变量说明＞  ::= ＜变量定义＞;{＜变量定义＞;}

分析：变量说明是由变量定义加上分号这个结构出现一次以上组成的。

样例：int i;

Float b,f[2];

19、＜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＞‘]’ )}

分析：变量定义是由类型标识符后面加上标识符或者标识符后面加上[＜无符号整数＞]结构组成，其中,(＜标识符＞|＜标识符＞’＜无符号整数＞] )这个结构可以出现一次以上。

样例： int a,b[4],c

Float f[2],d

20、＜类型标识符＞      ::=  int | float | char

分析：类型标识符是由int或float或char组成。

样例：int ，char

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

分析：有返回值函数定义是由声明头部加上(＜参数＞)这个结构后面加上{＜复合语句＞}结构组成的。

样例： int a(参数)

{

复合语句

}

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

分析：无返回值的函数定义是由void加上标识符加上(＜参数＞)这个结构后面加上{＜复合语句＞}结构组成的。

样例：void a(参数)

{

复合语句

}

23、＜复合语句＞   ::=  ［＜常量说明＞］［＜变量说明＞］＜语句列＞

分析：复合语句是由常量说明，变量说明以及语句列部分顺序组成，其中常量说明和变量说明部分可有可无。

样例： cons tint a= 0;

Int b;

语句列

24、＜参数＞    ::= ＜参数表＞

分析：参数是由参数表组成。

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

分析：参数表是由类型标识符加上标识符，后面加上若干次,＜类型标识符＞＜标识符＞结构或者由空组成。

从这里可以看出不能够传数组。

样例：int a,int b

Char c,float b

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

分析：主函数是由void main(){<复合语句>}这个结构组成。

样例：void main()

{

复合语句

}

27、＜表达式＞    ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞}

分析：表达式是由前面带有正号或者负号或者没有符号的<项>后面加上若干次加法运算符和<项>的结构组成。

样例：+ <项> - <项>

28、＜项＞     ::= ＜因子＞{＜乘法运算符＞＜因子＞}

分析：项是由因子后面加上若干次乘法运算符与因子的结构组成。

样例：<因子>\*<因子>

<因子>/<因子>

29、＜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＞‘)’

分析：因子是由标识符 或者 标识符后面加上中括号，中括号内为表达式结构 或者 整数 或者 实数 或者 字符 或者 有返回值的函数调用语句 或者 （<表达式>）组成。

样例：a

B[21]

C[2+d]

21

‘d’

0.1

A(2)

(2+d)

30、＜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＞;

分析：语句是由条件语句或者循环语句或者{<语句列>}或者有返回值的函数调用语句；或者无返回值的函数调用语句；或者赋值语句；或者读语句；或者写语句；或者；或者返回语句；组成。

样例：

条件语句

I = 0；

Conut（23）；

；

Return；

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

分析：赋值语句是由标识符加上等号表达式 或者 标识符[<表达式>] = 表达式组成。

样例：a = 2+3;

A[1] = 0;

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

分析：条件语句是由if加上(<条件>)语句，后面else<语句>可加可不加。

样例：if (a == 0)

{

C =0;

}

Else;

33、＜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

分析：条件是由表达式加上关系运算符加上表达式 或者 表达式组成。其中如果只包含表达式，则表达式为0时为假，否则为真。

样例: a == 1

b>a

c

d+5

34、＜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＞

分析：循环语句是由while（<条件>）加上语句 或者 for（标识符 = 表达式；条件；标识符= 标识符加或者减步长）后面加上语句得到。

样例：while(a == 0)

;

For(b =0;b<10;b=b+1)

;

35、＜步长＞    ::=  ＜非零数字＞｛＜数字＞｝

分析 ：步长是由非零数字后面加上若干位数字得到的，与无符号整数一致，这里的步长不允许是变量。

样例：200

23

36、＜有返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’

分析：有返回值的函数调用语句是由标识符(值参数表)组成。

样例： count(a,4)

37、＜无返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’

分析：无返回值的函数调用语句由标识符和(值参数表)组成，与有返回值的函数调用语句格式一样。

样例：send(c,3)

38、＜值参数表＞   ::= ＜表达式＞{,＜表达式＞}｜＜空＞

分析：值参数表是有表达式后面加上，<表达式>这个结构出现任意次组成 或者为空，所以这里不能够传地址或者数组。

样例 a+2,23,14

d

39、＜语句列＞   ::= ｛＜语句＞｝

分析：语句列是由语句出现若干次组成。

样例：a = 0;

B[1] = 4;

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

分析：读语句是由保留字scanf(标识符,＜标识符＞)组成，其中{，标识符}结构可以出现任意次。

这里就没有c语言的严格的类型定义了。  
样例：scanf(a,d,c[1])

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

分析：写语句是由printf保留字(<字符串>)或者printf(<表达式>)或者printf(<字符串>，<表达式>)组成。

样例：printf(1)

Printf(2+sa)

Printf(“sawda”)

Printf(“saws”,23+1)

42、＜返回语句＞   ::=  return[‘(’＜表达式＞‘)’]

分析：返回语句是由return加上或者不加上(<表达式>)组成的。

样例：return

Return (2)

Return (a+b)

1. **目标代码说明**

Pascal-S指令代码集（修改）：

"LDA",//0 加载变量地址到栈顶

"LOD",//1 加载变量到栈顶

"LDI",//2 加载整数到栈顶

"INT",//3 栈顶元素加上立即数

"JMP",//4 无条件跳转

"JPC",//5 有条件跳转，栈顶为0跳转

"CAL",//6 函数调用指令

"LDC",//7 加载字符到栈顶

"LDR",//8 加载实数到栈顶

"FLT",//9 整数转换为浮点数

"RED",//10 读指令//dx标识类型，1：整型，2：实型 3：字符型

"WRS",//11 写字符

"EXF",//12 函数结束指令

"LDT",//13 当前栈顶内容为地址，去当前栈顶地址处的值替换栈顶

"MUS",//14 堆栈顶取负

"WRR",//15 写实数

"STO",//16 把栈顶值存到次栈顶所在地址处

"EQR",//17 实数等于比较，等于就设为1，否则设0

"NER",//18 实数不等比较

"LSR",//19 实数小于比较

"LER",//20 实数小于等于比较

"GTR",//21 实数大于比较

"GER",//22 实数大于等于比较

"EQL",//23 整数等于比较

"NEQ",//24 整数不等比较

"LSS",//25 整数小于比较

"LER",//26 整数小于等于比较

"GRT",//27 整数大于比较

"GEQ",//28 整数大于等于比较

"ADD",//29 次栈顶加栈顶放在次栈顶（整数）

"SUB",//30 次栈顶减栈顶放在次栈顶（整数）

"ADR",//31 次栈顶加栈顶放在次栈顶（实数）

"SUR",//32 次栈顶减栈顶放在次栈顶（实数）

"MUL",//33 次栈顶乘栈顶放在次栈顶（整数）

"DIV",//34 次栈顶除栈顶放在次栈顶（整数）

"MUR",//35 次栈顶乘栈顶放在次栈顶（实数）

"DIR",//36 次栈顶除栈顶放在次栈顶（实数）

"WRW",//37 写整数

"MKS"//38 标记栈（函数调用前的初始化操作）

1. **详细设计**
2. **程序结构**

程序入口，进行一系列的类型定义，初始化符号表

读取一个单词，进行词法分析程序

错误处理

语法分析，不断读取单词，进行语义判断

生成目标代码

1. **类/方法/函数功能**

函数：

|  |  |  |
| --- | --- | --- |
| 函数名 | 函数作用 | 关键算法 |
| 词法分析： |  |  |
| Void nextc() | 获取下一个字符 | 如果是换行符或空格就再读取下一个，否则直接保存字符 |
| Void getsym() | 获取下一个单词 | 调用nextc函数并且判断分析当前单词类型 |
| 语法分析： |  |  |
| int sertable() | 在符号表中查找某个单词，没有就报错 | 从后往前查找标识符，查找到函数为止 |
| Void enter() | 把当前单词填入符号表，插入失败就报错 | 先遍历符号表，看是否有重复，再插入 |
| Void programme() | 程序分析程序 | 程序结构执行 |
| Void unsignedint() | 处理无符号整数 | 识别第一个数字不为0的整数 |
| Void quit\_table() | 翻译完函数后执行退表操作 | 处理完函数后把函数内常量以及变量退出符号表，仅保留函数名以及参数，并把参数名赋为空 |
| Void constantcon() | 常量声明分析程序 | While循环定义常量 |
| Void variablecon() | 变量声明分析程序 | While循环定义变量 |
| Void constantdef() | 常量定义分析程序 | 每个常量申明语句调用constantcon |
| Void variabledef() | 变量声明分析程序 | 每个变量申明语句调用variablecon |
| Void parameterlist() | 参数表处理程序 | While循环保存每个参数进符号表 |
| Void comstatement() | 复合语句处理程序 | 调用常量申明，变量申明，语句 |
| Void statement() | 语句处理程序 | 识别当前单词，调用不同的语句 |
| Void expression() | 表达式处理程序 | 循环调用项进行加减 |
| Void term() | 项处理程序 | 循环调用因子进行乘除 |
| Void factor() | 因子处理程序 | 识别单词进行各成分分析 |
| Void condition() | 条件处理程序 | 按条件格式进行成分分析 |
| Void ifstatement() | If语句处理程序 | 定位跳转位置，保存跳转指令下标 |
| Void whilestatement() | While语句处理程序 | 定位跳转位置，保存跳转指令下标 |
| Void forstatement() | For语句处理程序 | 定位跳转位置，保存跳转指令下标 |
| Void assignment(int i) | 赋值语句处理程序，传入被赋值变量在符号表中的下标 | 比较前后item类型是否一致，生成lda，lod，sto指令 |
| Void readstatement() | 读语句处理程序 | 读取数据到栈顶地址 |
| Void writestatement() | 写语句处理程序 | 把栈顶数据输出 |
| Void returnstatement() | 返回语句处理程序 | 比较返回值类型与ret类型是否一致 |
| Void call(int n) | 函数调用处理程序，传入函数在符号表中的下标 | 调用函数，用符号表中的函数起始位置进行跳转 |
| Void stat\_skip() | 语句中的错误处理跳读程序 | 语句的跳转进行错误结束符号集合构建 |
| Void statementlist() | 语句列处理程序 | 循环调用语句程序 |
| Void mainfunc() | Main函数处理程序 | Main函数，判断返回值为void，over=0 |
| Void un\_ret\_funcdef() | 无返回值函数处理程序 | 判断返回值为void，调用复合语句 |
| Void ret\_funcdef() | 有返回值函数处理程序 | 判断返回值类型是否一致，调用复合语句 |
| 目标代码生成 |  |  |
| Void emit(int f,int l,int d) | 生成Pcode指令 | 传入参数生成指令 |
| 解释执行程序 |  |  |
| Void interpret() | 解释执行程序 | 初始化堆栈，初始化各个关键寄存器，从第一条指令开始执行函数 |
| Void os\_error(int pc) | 运行时栈溢出错误，输出溢出指令序号 | 只会输出栈溢出错误 |
| 错误处理： |  |  |
| Void error(int n) | 根据错误代码生成错误信息 | Errmsg保存错误信息，传入错误信息对应的下标打印出错误语句 |
| Void skip() | 错误处理，跳到结束集合数组中的符号为止 | 错误处理，while循环一直读到在skipend数组中大于0的symbol为止 |

1. **调用依赖关系**

关系图如下：

语法分析程序

…一系列语句

复合语句

处理程序

返回一个单词

常量声明

词法分析程序

Stat\_skip()

变量声明

Getsym()

函数声明

返回一个字符

Nextc()

生成目标代码

Skip()

Error()

Main()

Interpret()

1. **符号表管理方案**

enum objtype {constant = 1,variable,funztion};

enum stitype {nottp = 1,intty,floatty,charty,arrayty};

typedef struct{ //符号表结构体建立

char name[ilong]; //符号的名字

enum objtype ty; //该符号所对应的类型（常量，变量和函数）

enum stitype typ; //数据类型，整型，浮点型，字符型和数组

int aindex; //数组信息向量表指针，否则为-1，若为函数则保存函数所占空间的大小

int addr; //如果是函数就记录该函数代码的起始地址，如果是整型常量就记录常量的值，否则记录常量在常量表的地址，如果是变量就记录变量的函数内的相对地址，如果是字符串就只想字符串的起始地址

int para; //如果符号是函数，则此变量记录参数个数，否则为-1

}table;

在扩展C0文法的编译器中，由于函数之间的变量无法相互引用以及语句块内无法定义常量或变量，所以符号表不需要level参数来进行区分语句块。

符号表内部属性说明：

|  |  |
| --- | --- |
| Name | 符号的名称 |
| Ty | 表示符号的类型。  常量为constant,变量为variable ,函数为function |
| typ | 表示符号具体的数据类型，整型为  Intty,浮点型为floatty,字符型为charty,如果是函数则记录函数的返回值类型，如果没有返回值，则为nottp。 |
| Aindex | 指向数组信息向量表指针，若为函数则保存函数所占空间的大小，否则为-1 |
| addr | 记录函数目标代码起始地址，整型常量的值，浮点型常量表的下表，字符串常量的起始下表以及变量的相对地址偏移 |
| Para | 记录函数的参数数目，否则为-1 |

符号表搜索方案：

把符号表0位置赋值为要检索的符号名，从符号表指针的所在位置开始，向下检索，找到的第一个同名的符号就返回符号表下标。

符号表更新操作：

当编译程序从一个函数体退出时，则把本函数局部变量全部退出符号表，仅保留函数名以及参数，并且把参数的名称赋值为空字符串。

符号表添加操作：

首先检索符号表中是否有该属性名，如果存在，就报错，否则添加到符号表指针当前位置，符号表指针加一。

1. **存储分配方案**

本程序中采用的栈式指令集，所有的指令操作都处在运行栈栈顶，所以仅仅采用栈来存储运行中所需的所有数据。

存储方式：栈式动态存储方式。

运行栈结构

|  |
| --- |
| 下一层函数局部变量区 |
| 函数调用参数 |
| 函数返回地址 |
| 上一层函数栈区基地址 |
| 函数返回值 |
| 此处为函数调用处 |
| 主函数区 |
| 全局变量区 |
| 栈底 |

由于在本程序中，所有的参数调用都是采用传值调用，所以在这里我们在调用函数的过程中，先为函数的返回值分配存储空间，再保存函数的返回地址，最后把本层函数的参数值放在堆栈栈顶，填入符号表。后面就开始下一层函数的局部变量存储。

1. **解释执行程序**

解释执行程序的初始化

解释执行程序中我们要为程序初始化一系列的堆栈全局变量以及主函数的局部变量分配内存为主函数的调用分配空间，然后再进行主函数的代码执行。

初始化代码：

int esp = 0; //栈顶位置

int abp = 0; //记录当前运行栈程序基地址

int pc = 1;

skipend[endsy] = 1;

order ic;

esp += ddx; //ddx为全局变量的大小

abp = esp;

esp += 2; //为主函数的3个参数分配空间（返回值，前栈帧adp以及返回地址，这里返回地址为0，对应指令数组的0位）

stack\_C[esp] = 0;

stack\_C[esp-1] = 0;

stack\_C[esp-2] = 0;

esp += mainsize-3; //mainsize为主函数大小（包括那3个参数，所以我们要减去），由语法分析程序编译计算产生

ic = code[pc]; //得到第一条指令内容

运行栈的结构为double类型的数组，为了实现操作数在栈顶的正确运行，我们需要保证栈的结构能够包含所有操作数的类型，由于其他类型的数据可以由double进行强制类型转化，所以我们这里设置double的数组来进行栈的操作（当然也可以采用联合UNION来实现栈元素的形式化）。

double stack\_C[stacksize];

关键寄存器

解释执行程序中包含几个关键寄存器

Pc:程序计数器，表示当前code代码的下标

Order：程序寄存器，保存当前指令

Abp：栈帧基址寄存器，表示当前栈帧基址对应的栈的下标

Esp：栈顶寄存器，保存栈顶的下标

Calabp[7]：参数基址寄存器组，用来保存函数调用参数处理时进入函数的栈帧基址，最多迭代7层

关键算法

每一个目标代码对应一个程序功能，具体的代码作用见上面的目标代码解释项目，仿照pascal解释执行程序格式。读取当前的目标代码，switch…case…来针对每一个目标代码执行特定的操作，实际的操作是通过硬件实现，先用程序语言模拟，具体的代码详见interpret程序段。

解释执行过程

While(pc != 0)

{

Order = code[pc++];

Switch (order.fct)

{

Case 0: …

Case 1: …

…

}

}

主函数的返回地址是0，pc初始化的值为1，从第一条指令开始运行，直到main返回进入code[0]，这时跳出while循环，得到最终程序运行结果，中间过程由switch case语句提供目标代码功能部分。

1. **目标代码生成方案**

Pcode代码数据结构：

typedef struct{

int fct;

int lev;

int dx;

}order;

order code[cmax];

int cx = 1;

因为扩展C0文法中并没有对层次有运用，因为函数之间不能调用相互的变量，所以lev位用来区分全局变量以及局部变量，0为全局变量，1为局部变量。创建Pcode代码数组，按照介绍的Pcode代码的含义对Pcode代码的解释，然后就根据操作生成对应的Pcode指令。Pcode代码数组的0下标处保留作为main函数的返回地址。同时也是程序结束标志。

生成代码函数：

void emit(int f,int l,int d)

{

if(cx == cmax)

error(46);

else

{

code[cx].fct = f;

code[cx].lev = l;

code[cx++].dx = d;

}

return;

}

函数传入三个参数，分别是指令的指令码，以及后两个参数，没有的话就传入0，code为目标代码数组，数组元素是order。

1. **出错处理**

错误类型代码以及错误信息解释如下：

|  |  |  |
| --- | --- | --- |
| 错误信息 | 错误类型代码 | 解释 |
| "ident too long" | 0 | 标识符太长 |
| "digit too long" | 1 | 数字过长 |
| "invalid char" | 2 | 不合法字符 |
| "without single quota" | 3 | 缺少一个单引号 |
| "string too long" | 4 | 字符串过长 |
| "repeat defined" | 5 | 标识符重复定义 |
| "too much symbol" | 6 | 符号表溢出 |
| "undefined word" | 7 | 未定义的单词引用 |
| "unsignedint error" | 8 | 无符号整数格式错误 |
| "without = " | 9 | 缺少等号 |
| "typedef inconsistent" | 10 | 定义类型不一致 |
| "constdef error" | 11 | 常量定义错误 |
| "without ;" | 12 | 缺少分号 |
| "not ident" | 13 | 此处应为标识符 |
| "without ]" | 14 | 缺少右中括号 |
| "withour )" | 15 | 缺少右小括号 |
| "no\_ret\_func not factor" | 16 | 无返回值函数不能作为因子 |
| "without [" | 17 | 缺少右中括号 |
| "array index not int" | 18 | 数组下标不为int类型 |
| "format error" | 20 | 格式错误 |
| "unknown factor" | 21 | 不合法的因子 |
| "type inconsistent" | 22 | 类型不一致 |
| "char can't be times ope" | 23 | Char类型不能作为乘积操作数 |
| "char can't be div ope" | 24 | Char类型不能做为除法操作数 |
| "without (" | 26 | 缺少左小括号 |
| "not variable no change" | 29 | 不是变量无法赋值 |
| "step on one ident" | 30 | For语句步长对一个操作数进行改变 |
| "ident not function" | 34 | 标识符应为函数 |
| "paranum inconsistent" | 35 | 参数数目类型不对应 |
| "ident not array" | 36 | 标识符应为数组 |
| "array can't be assigned" | 37 | 数组整体不能被赋值 |
| "without }" | 38 | 缺少右大括号 |
| "without {" | 41 | 缺少做大括号 |
| "without main func" | 44 | 缺少main函数 |
| "redundant code" | 45 | 多余的代码，不会被运行到 |
| "out of code array" | 46 | Code过长 |
| "return not inconsistent" | 47 | Return类型不一致 |
| "without returnstatement" | 48 | 有返回值函数没有return语句 |
| Too much array | 49 | 数组定义过多，数组信息向量表越界 |

出错处理方案：

首先存在一个结果符号集合

Int skipend[41];

41项分别对应程序所设定的41中symbol，如skipend[1]表示charsy，

Skipend[1]>0时表示charsy这个symbol在错误符号集合中，错误处理函数skip如下

Void skip()

{

while(skipend[sym] == 0)

getsym();

}

即程序一直会跳读到错误处理符号集合中的元素为止。

同时，错误结束符号集合是随着程序的运行而变化的。

比如，进入赋值语句时会执行skipend[semicolon]++;

就是把分号加入错误结束符号集合中，退赋值语句后再执行skipend[semicolon]--;

就把这个对应的分号结束符号取消。

如果出现明显的错误类型，可以给予改正，比如赋值语句中的=识别是==，就会报错并且执行后面部分而不进行跳读。否则就开始skip，跳过中间字符，知道碰到在终结符号集合skipend中的单词为止。

1. **操作说明**
2. **运行环境**

本工程文件是在codeblocks下编译完成的，请先完成codeblocks的环境配置工作，把本文件夹拷贝到codeblocks的workplace下即可，要保证文件路径中不包含中文名。

1. **操作步骤**

首先准备好要编译的源程序文件，直接运行本工程文件，控制台会输出please input the file path:的字样，这时输入源程序文件的完整路径，即可进行编译，编译完成之后会输出complete compiler的字样，如果源程序中包含词法错误（包括不合法的词语），语法错误（不合法的语句结构），语义错误（不合法的对数据的操作）等等，就会在控制台输出错误序列，错误序列中包含错误所在的行数，第几个字符等等，必须改正所有的错误，否则翻译所得的code代码不会存储到文件中。如果源程序中没有错误，那么就会直接编译把源程序翻译所得的code代码保存在文件夹下14061103\_code.txt中。同时在控制台中运行源程序所得到的code代码。在程序运行的过程中，只会产生一个报错，就是运行栈溢出，如果出现out of stack的字样，说明测试程序过于复杂，或者包含太多变量，太大的数组等等。还有一个重要的一点，本程序对于参数中包含函数调用的层数具有限制，即a(a(a(a(i))))这样的函数调用，规定最高的迭代层数不能超过7层。除此之外，程序运行结果会直接打印在控制台中，输入也通过控制台输入实现。

1. **测试报告**
2. **测试程序及测试结果**

**Test1：**

const int a = 5,b = 3;

int n;

int cur[30];

int fun[100];

int count(int choice,int start)

{

int i,num;

if(choice == 0)

{

num = 0;

for(i = 0;i<a;i = i +1)

{

num = num + start;

start = start+1;

}

}

else

{

num = 1;

for(i = 0;i<b;i= i+1)

{

num = num\*start;

start = start +1;

}

}

return (num);

}

void test\_add\_mul()

{

int result;

scanf(n);

result = count(0,n);

printf("add ",result);

result = count(1,n);

printf("mult ",result);

return;

}

void test\_arrayop()

{

int v,n,i,j,m,s,k;

scanf(v,n);

i = 0;

while(i<v)

{

scanf(m);

cur[i] = m;

i = i+1;

}

fun[0]=1;

i =0;

while(i<v)

{

s = cur[i];

j=s;

while(j<=n)

{

fun[j]= fun[j] +fun[j-cur[i]];

j = j+1;

}

i=i+1;

}

printf(fun[n]);

return;

}

void main()

{

test\_add\_mul();

test\_arrayop();

return;

}

输入：5

3 5 1 2 5

输出：add 35

Mult 210

4

**Test2:**

const int a = 0,b = -1;

const float c = --0.1,d = -+0003.;

const char j = '9';

int h,i[8],s;

float k,m;

char p[2],l [3];

int print\_i (int a)

{

printf(a);

return (0);

}

char print\_c(char a)

{

printf(a);

return('0');

}

float print\_f(float a)

{

printf(a);

return(0.0);

}

void opr (int a,char b,float c)

{

printf("int:a ",a);

printf("char:b ",b);

printf("float:c ",c);

printf(a+b);

printf("equal",b+a);

printf(a-b);

printf("opp",b-a);

printf(a+c);

printf("equal",c+a);

printf(a-c);

printf("opp",c-a);

}

void compare\_i(int a,int b)

{

if(a>b)

{

printf("a>b ");

printf(a);

printf(b);

}

if(a<b)

{

printf("a<b");

printf(a);

printf(b);

}

if(a == b)

{

printf("a=b");

printf(a);

printf(b);

}

}

void compare\_c(char a,char b)

{

if(a>b)

{

printf("a>b ");

printf(a);

printf(b);

}

if(a<b)

{

printf("a<b");

printf(a);

printf(b);

}

if(a == b)

{

printf("a=b");

printf(a);

printf(b);

}

}

void compare\_f(float a,float b)

{

if(a>b)

{

printf("a>b ");

printf(a);

printf(b);

}

if(a<b)

{

printf("a<b");

printf(a);

printf(b);

}

if(a == b)

{

printf("a=b");

printf(a);

printf(b);

}

}

void main()

{

printf("const int a",a);

h = -2; i[0]=100;

k = 0.09; m = -11.9;

p[1] = '9'; l[2] = 'a';

print\_i(h);

print\_c(p[1]);

print\_f(m);

opr(h,p[1],k);

compare\_i(h,i[0]);

compare\_c(p[1],l[2]);

compare\_f(k,m);

return;

}

输入：无

输出：const int a0

-2 9 -11.9

Int:a -2

Char:b 9

Float:c 0.09

55 equal 55

-59 opp 59

-1.91 equal -1.91

-2.09 opp 2.09

A<b -2 100

A<b 9 a

a>b 0.09 -11.9

**Test3:**

int recursion(int N)

{

if(n==0)

return (1);

else

return (n\*recursion(n-1));

}

int iteration(int n)

{

return (n+1);

}

void main()

{

int A;

scanf(a);

printf(recursion(a));

printf(iteration(iteration(iteration(iteration(recursion(a))))));

return;

}

输入：6

输出：720

724

**Test 4:**

int array[10];

int size;

int search(int n)

{

int i,j,m;

i = 0;

j = size;

m = (i+j)/2;

if(n <= array[m])

j = m - 1;

if(n>= array[m])

i = m + 1;

while(i <= j)

{

m = (i+j)/2;

if(n <= array[m])

j = m - 1;

if(n>= array[m])

i = m + 1;

}

if(i-1 > j)

return (m);

else

return (-1);

}

void search\_div2()

{

int i;

int n;

printf("watch the array:");

for(i = 0;i<size;i = i+1)

printf(array[i]);

printf("please input the number you want to search:");

scanf(n);

printf("the loc of the number is: ",search(n)+1);

return;

}

void init()

{

array[0] = 100;

array[1] = -89;

array[2] = 100;

array[3] = 9;

array[4] = -77;

array[5] = 4;

array[6] = -4;

array[7] = 12;

array[8] = 111;

array[9] = -111;

}

void qqsort(int left, int right)

{

int i;

int j;

int key;

int fi;

int fj;

i = left;

j = right;

key = array[left];

if(left >= right)

{

return ;

}

while(i < j)

{

fj = 0;

if(i<j) if(key>=array[j])fj = 1;

fi = 0;

if(i<j) if(key<array[i])fi = 1;

while(fj>=1) {

j = j - 1;

fj = 0;

if(i<j) if(key>=array[j])fj = 1;

}

array[i] = array[j];

while(fi>=1) {

i = i + 1;

fi = 0;

if(i<j) if(key<array[i])fi = 1;

}

array[j] = array[i];

}

array[i] = key;

qqsort(left, i - 1);

qqsort(i + 1, right);

}

void sort()

{

int i;

init();

printf("watch the array:");

for(i = 0;i<size;i=i+1)

printf(array[i]);

qqsort(0,9);

printf("after sort! ");

for(i = 0;i<size;i= i+1)

printf(array[i]);

}

void main()

{

int temp,i;

size = 10;

sort();

for(i = 0;i<size/2;i= i+1)

{

temp = array[i];

array[i] = array[size-i-1];

array[size-i-1]=temp;

}

search\_div2();

}

输入：-89

输出：watch the array:

100 -89 100 9 -77 4 -4 12 111 -111

After sort!

111 100 100 12 9 4 -4 -77 -89 -111

Watch the array:

-111 -89 -77 -4 4 9 12 100 100 111

Please input the number you want to search:(输入的-89)

The loc of the number is : 2

**Test 5:**

const int a = 3,b = 3;

const float c = 0.0,d = -1.0;

int high[10],low[10];

float result;

char first[20],second[20];

char j;

int count (int number)

{

int sum;

int i;

sum = 0;

if(number == 1)

{

for(i = 0;i<10;i= i+1)

sum = sum+first[i];

}

else

{

for(i = 0;i<10;i= i+1)

sum = sum+second[i];

}

return (sum);

}

char init(int n)

{

int i;

if(n != 2)

j = 'F';

else

{

for(i = 0;i<10;i= i+1)

first[i] = i;

for(i = -5;i<5;i = i+1)

second[i+5] = i;

}

return (j);

}

float accumlate(int n)

{

if(n <= 0)

return (0);

else

return (accumlate(n-1)+n);

}

int addi(int a,int b)

{

return (a+b);

}

int invalid()

{

int i;

i = 2;

i = addi(addi(3,addi(i,-2)),9);

return (i);

}

void test\_include()

{

int i;

int a,s;

int b[4];

a =0;

if(a ==0)

while(a)

a=1;

if(a ==0)

while(a == 0)

a =1;

printf("if while true1");

if(a!=0)

for(i =0;i<4;i = i+1)

b[i] = i;

if(b[3] ==3)

printf("if for true2");

while(b[3]>0)

if(b[3]>0)

b[3] = b[3] -1;

printf("while if true3");

i = 2;

a = 1;

while(i>0)

{

for (s =0;s<2;s=s+1)

a=a\*2;

i = i-1;

}

if(a == 16)

printf("while for true4");

s = 1;

for(i = 0;i<2;i= i+1)

{

if(i == i)

a = 2;

while(a>0)

{

s = s\*2;

a = a-1;

}

}

if(s == 16)

printf("for if true!5for while true!6");

}

void main()

{

int num,a;

float b;

char c;

j = 'T';

c = init(2);

if(c != 'T')

printf("error");

scanf(num);

if(num == 1)

a = count(1);

else

a = count(0);

printf(accumlate(a));

printf(invalid());

test\_include();

return;

}

输入：1

输出：1035.0

12

If while true1

If for true2

While if true3

While for true4

For if true5

For while true6

**ERRTest 1:**

在test1文件中变量定义删去分号int v,n,i,j,m,s,k

数组元素引用时删去右括号 fun[0=1;

输出：err 1:this program error in the line 42,9:without ;

err 2:this program error in the line 50,10:without ];

**ERRTest 2:**

在test2文件中printf语句中多加一个表达式 printf("const int a",a,a);

数组元素赋值时 i[]=100;

输出：err 1: this program error in the line 108,24:without )

Err 2: this program error in the line 109,12:unknown factor

Err 3: this program error in the line 109,12:array index not int

**ERRTest 3:**

函数参数缺少

Int函数没有返回值

int recursion(int)

{

if(n==0)

return;

else

return (n\*recursion(n-1));

}

int iteration(int n)

{

Int N;

return (n+1);

}

void main()

{

int a;

scanf(a);

printf(recursion(a));

printf(iteration(iteration(iteration(iteration(recursion(a))))));

return;

}

输出：err 1 :this program error in the line 1,18:not ident

err 2 :this program error in the line 3,5:undefined word

err 3 :this program error in the line 4,9:return not inconsistent

err 4 :this program error in the line 6,11:undefined word

err 5 :this program error in the line 6,27:without ;

err 6 :this program error in the line 10,6:repeat defined

err 7 :this program error in the line 16,20:paranum inconsistent

err 8 :this program error in the line 17,60:paranum inconsistent

**ERRTest 4:**

在test4文件中for赋值缺少等号 for(i 0;i<size/2;i= i+1)

函数调用缺少左括号 search\_div2);

输出：err 1 :this program error in the line 118,9:without =

err 2 :this program error in the line 118,10:unknown factor

err 3 :this program error in the line 118,10:type inconsistent

err 4 :this program error in the line 124,13:without (

**ERRTest 5:**

在test5中使用未定义变量 scanf(num);

if(num == 1)

float类型赋值给int int a; float b; char c; a = b;

输出：err 1 :this program error in the line 73,10:undefined word

err 2 :this program error in the line 73,10:not variable no change

err 3 :this program error in the line 74,7:undefined word

err 4 :this program error in the line 75,8:type inconsistent

1. **测试结果分析**

Test1主要测试对数的加减法以及乘除法，同时还包括一个货币问题的数组操作函数。

Test2 主要测试int，float，char之间的比较以及同种类型之间的比较以及加减法等等。

Test3主要是对递归以及函数调用还有函数调用的参数中引用函数调用等测试，程序较短。同时也检测标识符的大小写影响，测试3的错误程序中包含了定义一个大小的N以及小写的n，结果输出error：repeat defined表示标识符不区分大小写，大小写不同的标识符算一个变量。

Test4 主要是对于数组遍历，数组排序以及数组越界的测试，还包括二分法的数组查找。

Test5 主要测试for语句，if语句，if语句for语句while语句等各种嵌套。

基本可以覆盖文法中所有的语法分支。

1. **总结感想**

编译技术这个大作业是根据我们的能力设定的完成一个小型的编译器，在接触之前我们还完全不知道这个应该怎么实现，应该是什么结构，对于我们都十分的虚无缥缈，即使是上完了理论课，也只是从理论上掌握了这个大概是什么样，应该怎么样，但是实现但依然十分的困难。后来在阅读了pl0以及pasca的编译器之后，才初步的了解了这个大型机械的整体结构，于是出于一开始的模仿，我们开始了自己的扩展C0文法的编译器设计之路，虽然一开始的部分确实可以借鉴pascal，但是写到运行栈的部分，就感觉出明显的不同了，此时不能再一味的模仿，需要我们自己创造自己的模型，我们开始什么了解C语言的运行栈结构，并且根据自己的是实际情况进行些许的调整。

整个大作业的时间历时两个月左右。在这个过程中我们经历的很多的艰辛，同时也收获了很多，可以说对于我们设计较大的规模的工程或者体系来说都是一次不可多得的经验吧。