C0编译器开发文档

# 1 词法分析

关键字: const, int, void, if, else, while, main, return, printf, scanf

＜加法运算符＞ ::= +｜－

＜乘法运算符＞ ::= \* ｜／

＜关系运算符＞ ::= ＜｜＜＝｜＞｜＞＝｜!＝｜＝＝

＜字符＞ ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

＜数字＞ ::= 0｜＜非零数字＞

＜非零数字＞ ::= 1｜．．．｜9

＜专用符号＞ ::= (｜)｜{｜}｜，｜；｜＝

<字符串> ::= "｛<合法字符> }"

**表格中多余符号的解释：**

词法分析部分是在老师上传文法标准之前完成的，参考的是网上的一份C0扩充文法，所以其中会有char，：之类的字符。

|  |  |  |  |
| --- | --- | --- | --- |
| 单词名称 | 类别编码 | 记忆符 | 单词值 |
| Const | 1 | CONSTSY |  |
| Int | 2 | INTSY |  |
| Char | 3 | CHARSY |  |
| Void | 4 | VOIDSY |  |
| If | 5 | IFSY |  |
| Else | 6 | ELSSY |  |
| While | 7 | WHILESY |  |
| Scanf | 8 | SCANFSY |  |
| Printf | 9 | PRINTFSY |  |
| Return | 10 | RETURNSY |  |
| 标识符 | 11 | IDSY |  |
| 整常数 | 12 | INTAGER |  |
| < | 13 | MORESY |  |
| > | 14 | LESSSY |  |
| ( | 15 | LPARSY |  |
| ) | 16 | RPARSY |  |
| { | 17 | LLPARSY |  |
| } | 18 | LRPARSY |  |
| ‘ | 19 | APOSSY |  |
| “ | 20 | QUATASY |  |
| ; | 21 | SEMISY |  |
| + | 22 | PLUSSY |  |
| - | 23 | MINUSSY |  |
| \* | 24 | MULSY |  |
| / | 25 | DIVISY |  |
| , | 26 | COMMASY |  |
| = | 27 | ASSIGNSY |  |
| : | 28 | COLONSY |  |
| >= | 29 | MOESY |  |
| <= | 30 | LOESY |  |
| == | 31 | EQUSY |  |
| != | 32 | UNEQUSY |  |
| 字符串 | 50 | STRINGSY |  |
| Main | 0 |  |  |
|  |  |  |  |



程序流程图如上图所示。

# 2 语法分析



























# 3 代码生成

代码生成部分最后的结果与语法分析中的流程图区别较大，因为要处理很多产生式FIRST集相交的问题，所以在实际编写的过程中将许多小的函数合并为了大的函数。

#### 变量说明

struct Charactertable

{

char name[MAX\_SIZE\_OF\_TOKEN];

int lev, kind, addr, type;

int num;

};

符号表，name是名字，lev是所在层次，kind是符号种类（分为常量，变量，有返回函数，无返回函数），num存储常量的值和函数的参数个数，type由于只有int和void，还综合在kind里，最后没用上。

static struct Charactertable ctbl[MAX\_SIZE\_OF\_CHARACTERTABLE];

符号表。

static int numOfCtbl;

符号个数。

static int numOfVarInLev[MAX\_SIZE\_OF\_CHARACTERTABLE];

每层变量个数，用于声明变量时计数。

struct oper

{

int op;

int lev;

int x;

};

Pcode的结构。

static struct oper pcode[MAX\_SIZE\_OF\_PCODE];

Pcode栈。

static int numOfPcode;

Pcode数量。

static char mystring[MAX\_SIZE\_OF\_STRING];

存放程序的字符串。

static char token[MAX\_SIZE\_OF\_TOKEN];

词法分析得到的结果值。

static char char\_pointed;

当前指针指向字符。

static int lev;

当前层次。

static int token\_p;

指针，用于指出分析的位置。

static int num;

存放词法分析得到的数字。

static int symbol;

词法分析得到的符号编号。

static int errorline;

当前行数。

static int errorchar;

当前行字符数。

static int errorflag;

用于判断是否输出Pcode

static const char \*reserver\_ls[] = {

"main",

"const",

"int",

"char",

"void",

"if",

"else",

"while",

"scanf",

"printf",

"return",

};

保留字。

static const char \*operation[] = {

"LIT",

"OPR",

"LOD",

"STO",

"CAL",

"INT",

"JMP",

"JPC",

"RED",

"WRT"

};

Pcode操作。

static const char \*error\_msg[] = {

"invalid symbol",

"lacking semicolon",

"invalid identifier",

"invalid declaration",

"lacking const",

"lacking identifier",

"lacking '='",

"lacking integer",

"double decleration",

"character table overflow",

"lacking '('",

"lacking 'int'",

"lacking ')'",

"lacking '{'",

"lacking '}'",

"invalid factor",

"wrong kind identifier",

"undeclared identifier",

"wrong number of parameters",

"invalid judgement",

"lacking statement",

"lacking ','",

"pcode overflow"

};

出错信息。

#### 函数说明

static void getchar1();

指针向前推进一个字符。

static void clearToken();

清理token

static bool isSpace();

判断符号是否是空格，tab，和换行

static bool isnum();

判断是否是数字

static bool isletter();

判断是否是字母和下划线

static bool ismore();

判断是否是大于号

static bool isless();

判断是否是小于号

static bool isLpar();

判断是否是左括号

static bool isRpar();

判断是否是右括号

static bool islLpar();

判断是否是左大括号

static bool islRpar();

判断是否是右大括号

static bool isquata();

判断是否是引号

static bool issemi();

判断是否是分号

static bool isplus();

判断是否是加号

static bool isminus();

判断是否是减号

static bool ismul();

判断是否是乘号

static bool isdivi();

判断是否是除号

static bool iscomma();

判断是否是逗号

static bool isassign();

判断是否是赋值符号

static bool iscolon();

判断是否是单引号

static bool ismoe();

判断是否是大于等于号

static bool isloe();

判断是否是小于等于号

static bool isequ();

判断是否是双等于号

static bool isunequ();

判断是否是不等于号

static int isreserver();

判断是否是保留字，如果是，返回相应编号，如果不是，返回-1

static void addToToken();

将当前字符加入token

static void retract();

指针回退一个字符

static int getsym();

词法分析程序，获取一个合法的单词

static void error(int msg\_id, int cx);

出错处理程序

static int progress();

语法，语义分析主程序

static int constinstr();

常量声明

static int constdef(char \*name, int \*count);

常量定义

static int enter(char const \*name, int type, int kind, int count);

填符号表

static int funcid(char \*name);

标识符处理函数

static int funcinteger(int \*count);

整数处理函数

static int funcwithreturn(char const \*name);

实际作用和名称不一样，用于处理void，int类型函数，包括main函数

static int varinstr(char \*name);

变量声明

static int statementseq();

语句序列

static int expr();

表达式

static int term();

项

static int factor();

因子

static int funccall(int funcinctbl);

函数调用

static int ifstatement();

判断语句

static int whilestatement();

while循环语句

static int assignstatement();

赋值语句

static int returnstatement();

返回语句

static int readstatement();

读语句

static int writestatement();

写语句

static int funcif();

条件语句

static int lookup(char \*name);

查符号表，返回查找符号在符号表中的序号

static int gen(int op, int lev, int x);

生成一条Pcode

static int pushdir(int x);

生成一条LIT语句

static int pushbyaddr(int lev, int x);

生成一条LOD语句

static int add();

生成加法Pcode语句

static int subt();

生成减法Pcode语句

static int mul();

生成乘法Pcode语句

static int div();

生成除法Pcode语句

static int sto(int lev, int x);

生成存储Pcode语句

static int reverse();

生成栈顶取反Pcode语句

static int jmp(int x);

生成无条件跳转Pcode语句

static int jmc(int x);

生成有条件跳转Pcode语句

static int wrt();

生成WRT Pcode语句

static int backout();

将不用的变量从符号表中退出

# 4. 测试用例

A.该例测试基本的变量声明，赋值语句，读语句，写语句，while循环语句

int a,b;

void main()

{

int x, y;

a = 3;

scanf(x);

while(a!=x)

{

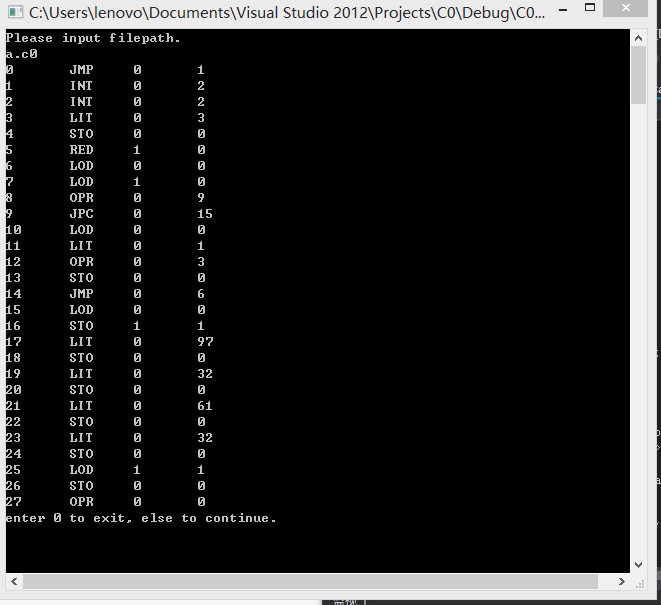
a = a - 1;

}

y = a;

printf("a = ",y);

}

B. 该例测试了函数声明，函数调用，if语句

这个例子的运行结果说明了一个经典的错误，swap的参数是值类型时无法完成交换的功能。

int swap(int a, int b)

{

int tmp;

tmp = a;

a = b;

b = tmp;

return (1);

}

int main()

{

int x, y;

x = 1;

y = 0;

swap(x, y);

if(x == 0)

{

printf("x =", y);

}

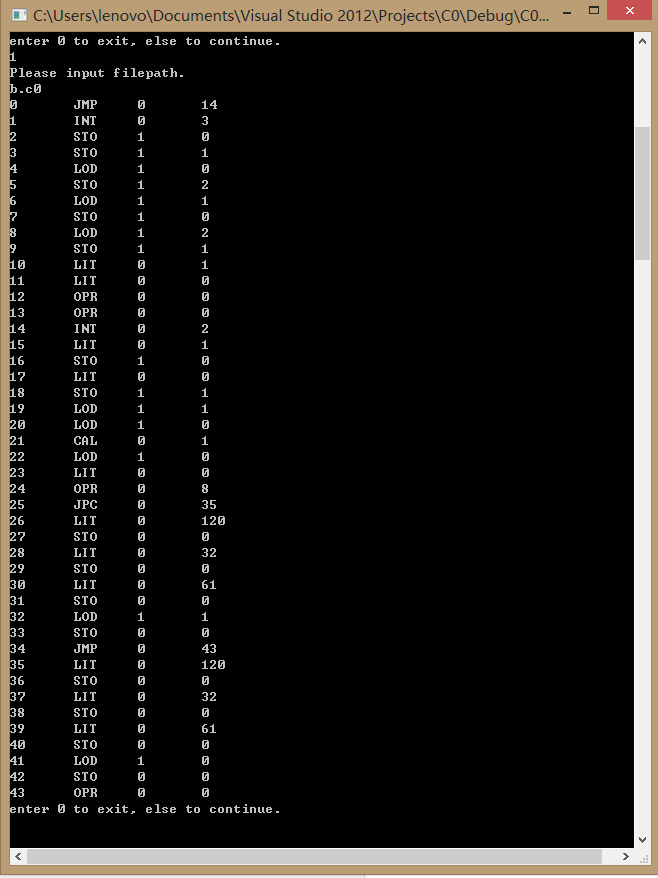
else

{

printf("x =", x);

}

}



1. 该例测试了常量声明调用，void函数调用，以及验证处于不同作用域的同名函数是可以

const q = 10;

int a, b, c;

void func1(int b, int c)

{

b = a/c;

c = b\*c;

}

void main(int b)

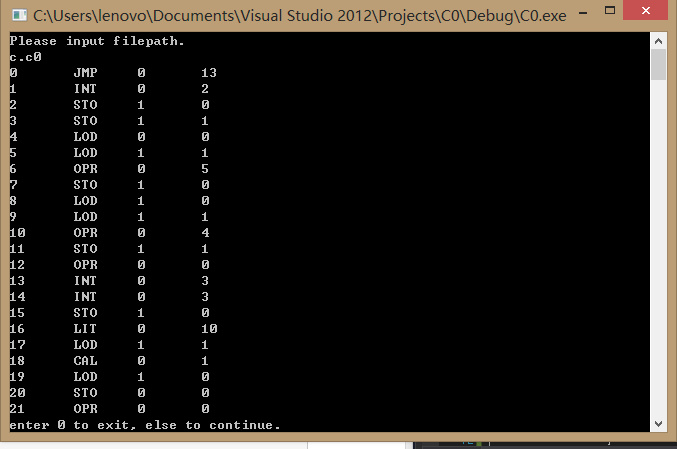
{

int x, y;

func1(x, q);

printf(b);

}



1. 该例声明了多个函数，并且使用同名参数，并加入了其他各种语句，验证编译器的可靠性。

int add(int n)

{

int sum, x;

while (n!= 0)

{

scanf(x);

sum = sum + x;

n = n - 1;

}

return (sum);

}

int subt(int n)

{

int sum, x;

while (n!= 0)

{

scanf(x);

sum = sum - x;

n = n - 1;

}

return (sum);

}

void main()

{

int a, b;

a = 5;

b = a;

if(a == b)

{

add(a);

subt(b);

}

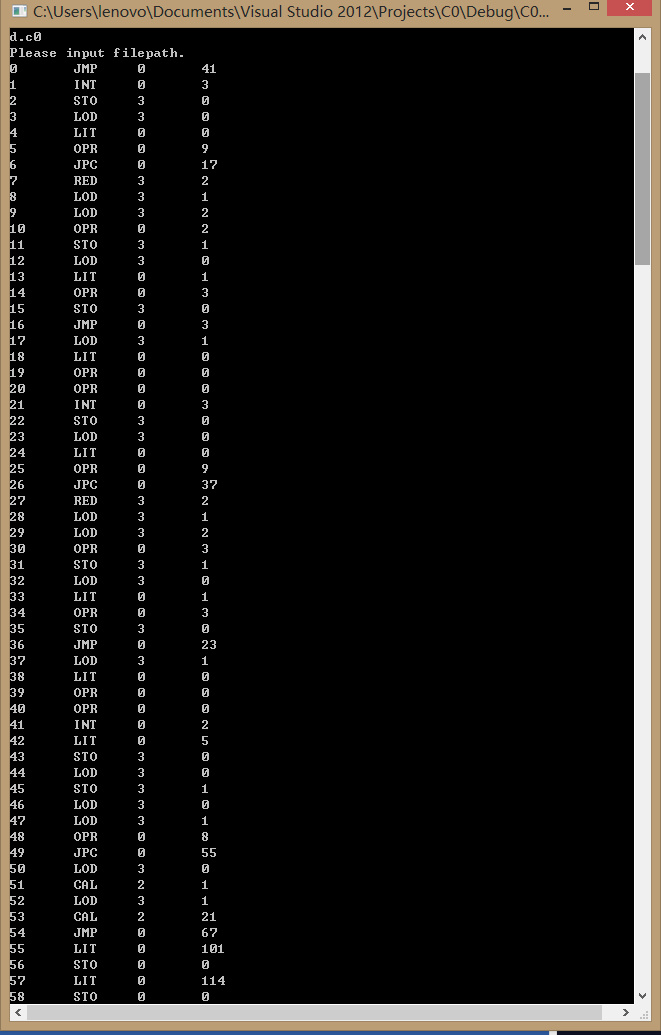
else

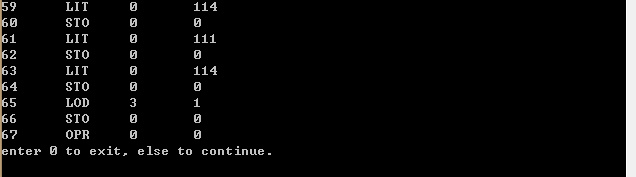
{

printf("error", b);

}

}





1. 将B改为错误的用例。例子中声明tmp时少了分号，所以tmp声明失败，和调用tmp的语句都报错。另外，声明了两次swap，出错分析判定为多重声明。

int swap(int a, int b)

{

int tmp

tmp = a;

a = b;

b = tmp;

return (1);

}

int swap(int a, int b){}

int main()

{

int x, y;

x = 1;

y = 0;

swap(x, y);

if(x == 0

{

printf("x =", y) ;

}

else

{

printf("x =", x);

}

}

