**电子科技大学 信息与软件工程 学院**

**标 准 实 验 报 告**

**（实验）课程名称 编译技术**

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：程阳 学 号：2012223010013 指导教师：周尔强**

**实验地点：信息与软件工程学院实验中心306 实验时间：2013.11.2**

**一、实验室名称：通用计算机软件实验室**

**二、实验项目名称： 递归下降分析法**

**三、实验学时：4学时**

**四、实验原理：**

符号表

单词

递归下降

分析法

程序

单词表示的

源程序

出错处理

递归下降分析法的实现方案

递归下降分析法的原理是利用函数之间的递归调用模拟语法树自上而下的构造过程，具体实现方法概述如下：

1）每个非终结符对应一个解析函数；

2）产生式右侧为该产生式左侧非终结符所对应解析函数的“函数体”；

3）产生式右侧终结符对应从输入串中“消耗”该终结符的操作；

4）产生式中的‘|’对应函数体中的“if-else”语句；

5）对于扩展的BNF文法，产生式中的‘{}’对应函数体中的“while”语句。

**五、实验目的：**

理解语法分析在编译程序中的作用，理解并掌握递归下降分析法的主要原理，理解该分析方法对文法的要求，熟练掌握文法改造算法（消除左递归和消除公共左因子），掌握递归下降分析程序的构造方法。

**六、实验内容：**

用C语言（或C++）对一个简单语言构造一个递归下降分析程序，具体实验内容包括：

1. 检查所处理语言的文法是否满足递归下降算法要求，如不满足，对其进行改造；
2. 根据递归下降算法编写所处理语言的递归下降程序；
3. 调用词法分析器，获取测试程序的词法记号；
4. 将词法分析后的符号表输出到文件symtab1.txt；
5. 对测试程序的词法记号进行分析、检查，输出分析结果（接受或拒绝），并将语法分析所得的错误信息输出到文件error.txt；
6. 将语法分析后的符号表输出到文件symtab2.txt；
7. 试将详细的分析步骤，包括每一步使用的产生式、已分析过的串、当前分析字符、剩余串输出到detail.txt（选做）。

**七、实验器材（设备、元器件）：**

硬件要求：pc机，CPU PII 以上，64M 内存，100M 硬盘空间即可。

软件要求：Windows 7/XP/2003等，包括C编译器的IDE。

1. **实验步骤、实验编程与运行结果：**

**源代码：**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define MAX\_ID\_LENGTH 256

#define MAX\_TABLE\_SIZE 2048

#define SUCCESS 1

typedef enum token

{

T\_0=0,

T\_1=1,

T\_2=2,

T\_INT=3,

T\_IF=4,

T\_ELSE=5,

T\_WHILE=6,

T\_FOR=7,

T\_READ=8,

T\_WRITE=9,

T\_BEGIN=10,

T\_FOUNCTION=11,

T\_END=12,

T\_THEN=13,

T\_ADD=14,

T\_SUB=15,

T\_MUL=16,

T\_DIV=17,

T\_L=18,

T\_LE=19,

T\_G=20,

T\_GE=21,

T\_NE=22,

T\_E=23,

T\_ASSIGN=24,

T\_LPAR=25,

T\_RPAR=26,

T\_COM=27,

T\_SEM=28,

T\_RETURN=29,

T\_ERROR=30,

T\_SYMBOL=31,

T\_CONSTANT=32

} Token;

typedef struct LEX

{

Token read;

struct LEX \*next;

size\_t line;

}MYLEX;

#ifndef SUCCESS

#define SUCCESS 1

#endif

#endif // HEAD\_H

MYLEX lex\_head,error\_temp;

MYLEX\* current;

Token error\_TOKEN=T\_0;

/\*初始化信息\*/

/\*函数申明\*/

int item(void);

int Block(void);

int Factor(void);

int Variable(void);

int Function(void);

int Constant(void);

int Parameter (void );

int Identifier (void);

int Main\_Program(void);

int Function\_call(void);

void emergency\_exit(void);

int Read\_statement(void);

int Write\_statement(void);

int Execute\_statement(void);

MYLEX\* Advance(MYLEX\* read);

MYLEX\* Read\_file(MYLEX\* read);

int Relational\_operators(void);

int Assignment\_statement(void);

int Variable\_declaration(void);

int Function\_declaration(void);

int match(Token read,Token aim);

int Conditional\_statement(void);

int Arithmetic\_expression(void);

int Declarative\_statement(void);

int Conditional\_expression(void);

int Execute\_statement\_table(void);

int error(MYLEX \*current,Token aim);

int Declarative\_statement\_table(void);

void emergency\_exit(void)

{

MYLEX\* EXIT\_TEEMP=lex\_head.next;

current=EXIT\_TEEMP;

while(!current)

{

current=EXIT\_TEEMP->next;

free(EXIT\_TEEMP);

}

exit(0);

}

int error(MYLEX \*current,Token aim)

{

FILE \*error\_file=fopen("error.txt","a");

if(!error\_file)

{

perror("can`t open file \n");

}

if(error\_temp.line<current->line)

{

error\_temp=\*current;

error\_TOKEN=aim;

fprintf(error\_file,"error in line:%d\tTOKEN:%d\texpeced Token is :%d\n",

error\_temp.line,error\_temp.read,error\_TOKEN);

}

fclose(error\_file);

return SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Read\_file

// FullName: Read\_file

// Access: public

// Returns: int

// Qualifier:读文件

// Parameter: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MYLEX\* Read\_file(MYLEX\* current)

{

current->line=0;

current->next=NULL;

current->read=T\_0;

MYLEX\* Temp;

Temp=current;

FILE\* READ\_FILE=fopen("GRAMER.txt","r");

if(!READ\_FILE)

{

printf("can`t load file GRAMER.txt, a file like GRAMER.txt is expectd here \n");

exit(0);

}

while(!feof(READ\_FILE))

{

Temp->next=(MYLEX\*)malloc(sizeof(MYLEX));

Temp=Temp->next;

if(!Temp)

{

printf("can`t malloc a space for this program!\n");

emergency\_exit();

}

Temp->line=0;

Temp->next=NULL;

Temp->read=T\_0;

fscanf(READ\_FILE,"%d\t%d\n",&(Temp->line),&(Temp->read));

}

fclose(READ\_FILE);

return current->next;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Advance

// FullName: Advance

// Access: public

// Returns: Token

// Qualifier: 前进移向下一个TOKEN

// Parameter: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MYLEX\* Advance(MYLEX\* current)

{

FILE\*EAT\_FILE=fopen("symtab2.txt","a");

FILE\* LEFT\_FILE=fopen("detail.txt","a");

if((!EAT\_FILE)||(!LEFT\_FILE))

{

perror("can`t open file");

emergency\_exit();

}

MYLEX\* Temp=current->next;

printf("you have eat the one is :%3d\n\n follow is \n",current->read);

fprintf(EAT\_FILE,

"\nyou have eat the one is line:%3d\tToken:%3d\n",

current->line,

current->read);

fprintf(LEFT\_FILE,

"\n\ncurrent eat the one is line:%3d\tToken:%3d\nfollow is\n",

current->line,

current->read);

for(Temp; Temp!=NULL; Temp=Temp->next)

{

printf("line:%3d \t token=%3d\n",

Temp->line,Temp->read);

fprintf(LEFT\_FILE,"line:%3d \t token=%3d\n",

Temp->line,Temp->read);

}

fclose(EAT\_FILE);

fclose(LEFT\_FILE);

if(current==NULL)

{

printf("all work is to be done!,there is an error in your project!\n ");

emergency\_exit();

}

return current->next;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: main

// FullName: main

// Access: public

// Returns: int

// Qualifier: 主程序

// Parameter: int agrc

// Parameter: char \* agrv[]

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main(int agrc, char\*agrv[])

{

error\_temp.line=0;

error\_temp.next=NULL;

error\_temp.read=T\_0;

printf("\t编译原理实验二\n\t2012223010013\_程阳\n\tNEWPLAN\n");

current=Read\_file(&lex\_head);

system("pause");

if(Main\_Program())

{

printf("\n\ncongratulation\tsuccessfully!\n\n");

system("del error.txt");

}

else

{

printf(" sorry you have failed!\n");

printf("error in line:%d\tTOKEN:%d\texpeced Token is :%d\n",

error\_temp.line,error\_temp.read,error\_TOKEN);

}

system("pause");

emergency\_exit();/\*释放空间，函数返回\*/

return 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: match

// FullName: match

// Access: public

// Returns: int

// Qualifier: 匹配

// Parameter: Token read

// Parameter: Token aim

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int match(Token read ,Token aim)

{

if(read==aim)

return SUCCESS;

error(current,aim);

return !SUCCESS;

// return read==aim? SUCCESS : !SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Main\_Program

// FullName: Main\_Program

// Access: public

// Returns: int

// Qualifier: 开始主程序

// Parameter: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Main\_Program(void)

{

if(!Block())

return !SUCCESS;

return SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Block

// FullName: Block

// Access: public

// Returns: int

// Qualifier:/\*分程序\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Block(void)

{

MYLEX\* Temp=current;

if(!match(current->read,T\_BEGIN))

{

current=Temp;

return !SUCCESS;

}

current= Advance(current);

if(!Declarative\_statement\_table())

{

current=Temp;

return !SUCCESS;

}

if(!match(current->read,T\_SEM))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

if(!Execute\_statement\_table())

{

current=Temp;

return !SUCCESS;

}

if(!match(current->read,T\_END))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

return SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Declarative\_statement\_table

// FullName: Declarative\_statement\_table

// Access: public

// Returns: int

// Qualifier:/\*说明语句表\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Declarative\_statement\_table()

{

MYLEX\* Temp=current;

if(!Declarative\_statement())

{

current=Temp;

return !SUCCESS;

}

/\*采用扩展的BNF方法消除左递归\*/

for(;;)

{

if(!match(current->read,T\_SEM))

break;

Temp=current;

current =Advance(current);

if(!Declarative\_statement())

{

current=Temp;

break;

}

}

return SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Declarative\_statement

// FullName: Declarative\_statement

// Access: public

// Returns: int

// Qualifier:/\*说明语句\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Declarative\_statement(void)

{

MYLEX\* Temp=current;

if(Variable\_declaration())

{

return SUCCESS;

}

else

{

current=Temp;

}

if(Function\_declaration())

return SUCCESS;

current=Temp;

return !SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Execute\_statement\_table

// FullName: Execute\_statement\_table

// Access: public

// Returns: int

// Qualifier:/\*执行语句表\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Execute\_statement\_table(void)

{

MYLEX\* Temp=current;

if(!Execute\_statement())

{

current=Temp;

return !SUCCESS;

}

for(;;)

{

if(!match(current->read,T\_SEM))

break;

Temp=current;

current=Advance(current);

if(!Execute\_statement())

{

current=Temp;

break;

}

}

return SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Execute\_statement

// FullName: Execute\_statement

// Access: public

// Returns: int

// Qualifier:/\*执行语句\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Execute\_statement(void)

{

MYLEX\* Temp=current;

if(Read\_statement())

{

return SUCCESS;

}

else

{

current=Temp;

}

if(Write\_statement())

{

return SUCCESS;

}

else

{

current=Temp;

}

if(Conditional\_statement())

{

return SUCCESS;

}

else

{

current=Temp;

}

if(Assignment\_statement())

{

return SUCCESS;

}

current=Temp;

return !SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Variable\_declaration

// FullName: Variable\_declaration

// Access: public

// Returns: int

// Qualifier:/\*变量说明\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Variable\_declaration(void)

{

MYLEX\* Temp=current;

if(!match(current->read , T\_INT))

{

current=Temp;

return !SUCCESS;

}

current =Advance(current);

if(Variable())

return SUCCESS;

current=Temp;

return !SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Function\_declaration

// FullName: Function\_declaration

// Access: public

// Returns: int

// Qualifier:/\*函数说明\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Function\_declaration(void)

{

MYLEX\* Temp=current;

if(!match(current->read, T\_INT))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

if(!match(current->read, T\_FOUNCTION))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

if(!Identifier())

{

current=Temp;

return !SUCCESS;

}

if(!match(current->read, T\_LPAR))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

if(!Parameter())

{

current=Temp;

return !SUCCESS;

}

if(!match(current->read, T\_RPAR))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

if(!match(current->read,T\_SEM))

{

current=Temp;

return !SUCCESS;

}

current =Advance(current);

if(!Function())

{

current=Temp;

return !SUCCESS;

}

return SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Variable

// FullName: Variable

// Access: public

// Returns: int

// Qualifier:/\*变量\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Variable(void)

{

MYLEX\* Temp=current;

if(! match(current->read,T\_SYMBOL))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

return SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Identifier

// FullName: Identifier

// Access: public

// Returns: int

// Qualifier:/\*标识符\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Identifier (void)

{

MYLEX\* Temp=current;

if(! match(current->read,T\_SYMBOL))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

return SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Parameter

// FullName: Parameter

// Access: public

// Returns: int

// Qualifier:/\*参数\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Parameter (void)

{

MYLEX\* Temp=current;

if(!Variable())

{

current=Temp;

return !SUCCESS;

}

return SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Function

// FullName: Function

// Access: public

// Returns: int

// Qualifier:/\*函数体\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Function(void)

{

MYLEX\* Temp=current;

if(!match(current->read, T\_BEGIN))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

if(!Declarative\_statement\_table())

{

current=Temp;

return !SUCCESS;

}

if(!match(current->read, T\_SEM))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

if(!Execute\_statement\_table())

{

current=Temp;

return !SUCCESS;

}

if(!match(current->read,T\_END))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

return SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Read\_statement

// FullName: Read\_statement

// Access: public

// Returns: int

// Qualifier:/\*读语句\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Read\_statement(void)

{

MYLEX\* Temp=current;

if(!match(current->read,T\_READ))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

if(!match(current->read,T\_LPAR))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

if(!Variable())

{

current=Temp;

return !SUCCESS;

}

if(!match(current->read,T\_RPAR))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

return SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Write\_statement

// FullName: Write\_statement

// Access: public

// Returns: int

// Qualifier:/\*写语句\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Write\_statement(void)

{

MYLEX\* Temp=current;

if(!match(current->read,T\_WRITE))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

if(!match(current->read,T\_LPAR))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

if(!Variable())

{

current=Temp;

return !SUCCESS;

}

if(!match(current->read,T\_RPAR))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

printf("current to be got is:line=%d\t %d\n",current->line,current->read);

return SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Assignment\_statement

// FullName: Assignment\_statement

// Access: public

// Returns: int

// Qualifier:/\*赋值语句\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Assignment\_statement(void)

{

MYLEX\* Temp=current;

if(!Variable())

{

current=Temp;

return !SUCCESS;

}

if(!match(current->read,T\_ASSIGN))

{

current=Temp;

return !SUCCESS;

}

current =Advance(current);

if(Arithmetic\_expression())

return SUCCESS;

current=Temp;

return !SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Conditional\_statement

// FullName: Conditional\_statement

// Access: public

// Returns: int

// Qualifier:/\*条件语句\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Conditional\_statement(void)

{

MYLEX\* Temp=current;

if(!match(current->read,T\_IF))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

if(!Conditional\_expression())

{

current=Temp;

return !SUCCESS;

}

if(!match(current->read,T\_THEN))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

if(!Execute\_statement())

{

current=Temp;

return !SUCCESS;

}

if(!match(current->read,T\_ELSE))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

if(Execute\_statement())

return SUCCESS;

current=Temp;

return !SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Arithmetic\_expression

// FullName: Arithmetic\_expression

// Access: public

// Returns: int

// Qualifier:/\*算术表达式\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Arithmetic\_expression(void)

{

MYLEX\* Temp=current;

if(!item())

{

current=Temp;

return !SUCCESS;

}

for(;;)

{

if(!match(current->read,T\_SUB))

break;

Temp=current;

current=Advance(current);

if(!item())

{

current=Temp;

break;

}

}

return SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: item

// FullName: item

// Access: public

// Returns: int

// Qualifier:/\*项\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int item(void)

{

MYLEX\* Temp=current;

if(!Factor())

{

current=Temp;

return !SUCCESS;

}

for(;;)

{

if(!match(current->read,T\_MUL))

break;

Temp=current;

current=Advance(current);

if(!Factor())

{

current=Temp;

break;

}

}

return SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Constant

// FullName: Constant

// Access: public

// Returns: int

// Qualifier:/\*常数\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Constant(void)

{

MYLEX\* Temp=current;

if(!match(current->read,T\_CONSTANT))

{

return !SUCCESS;

}

current=Advance(current);

return SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Conditional\_expression

// FullName: Conditional\_expression

// Access: public

// Returns: int

// Qualifier:/\*条件表达式\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Conditional\_expression(void)

{

MYLEX\* Temp=current;

if(!Arithmetic\_expression())

{

current=Temp;

return !SUCCESS;

}

printf("line = %d,token=%d\n");

if(!Relational\_operators())

{

current=Temp;

return !SUCCESS;

}

if(!Arithmetic\_expression())

{

current=Temp;

return !SUCCESS;

}

return SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Relational\_operators

// FullName: Relational\_operators

// Access: public

// Returns: int

// Qualifier:/\*关系运算符\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Relational\_operators(void)

{

if(match(current->read,T\_L))

{

current=Advance(current);

return SUCCESS;

}

if(match(current->read,T\_LE))

{

current=Advance(current);

return SUCCESS;

}

if(match(current->read,T\_G))

{

current=Advance(current);

return SUCCESS;

}

if(match(current->read,T\_GE))

{

current=Advance(current);

return SUCCESS;

}

if(match(current->read,T\_E))

{

current=Advance(current);

return SUCCESS;

}

if(match(current->read,T\_NE))

{

current=Advance(current);

return SUCCESS;

}

return !SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Factor

// FullName: Factor

// Access: public

// Returns: int

// Qualifier:/\*因子\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Factor(void)

{

MYLEX\* Temp=current;

if(Function\_call())

{

return SUCCESS;

}

else

{

current=Temp;

}

if(Variable())

{

return SUCCESS;

}

else

{

current=Temp;

}

if(Constant())

{

return SUCCESS;

}

current=Temp;

return !SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Function\_call

// FullName: Function\_call

// Access: public

// Returns: int

// Qualifier:/\*函数调用\*/

// Parameter: Token read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Function\_call(void)

{

MYLEX\* Temp=current;

if(!Identifier())

{

current=Temp;

return !SUCCESS;

}

if(!match(current->read,T\_LPAR))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

if( !Arithmetic\_expression())

{

current=Temp;

return !SUCCESS;

}

if(!match(current->read,T\_RPAR))

{

current=Temp;

return !SUCCESS;

}

current=Advance(current);

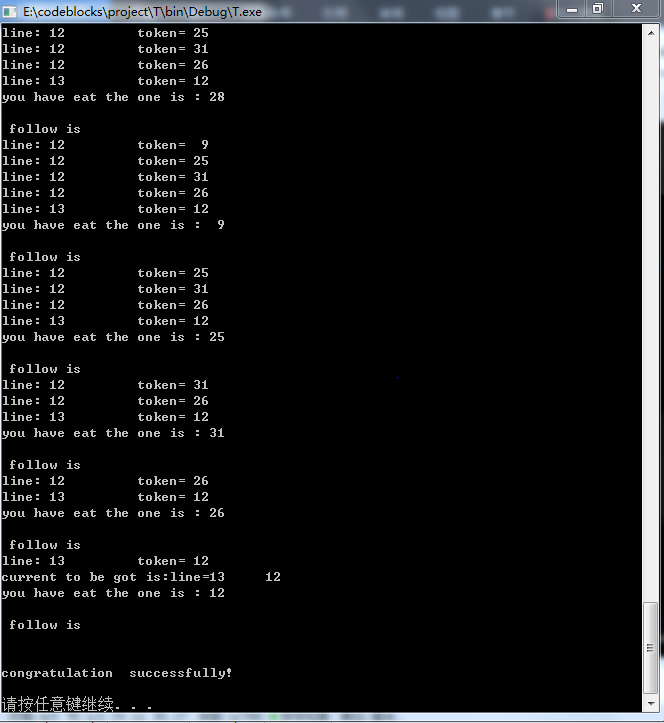
return SUCCESS;

}

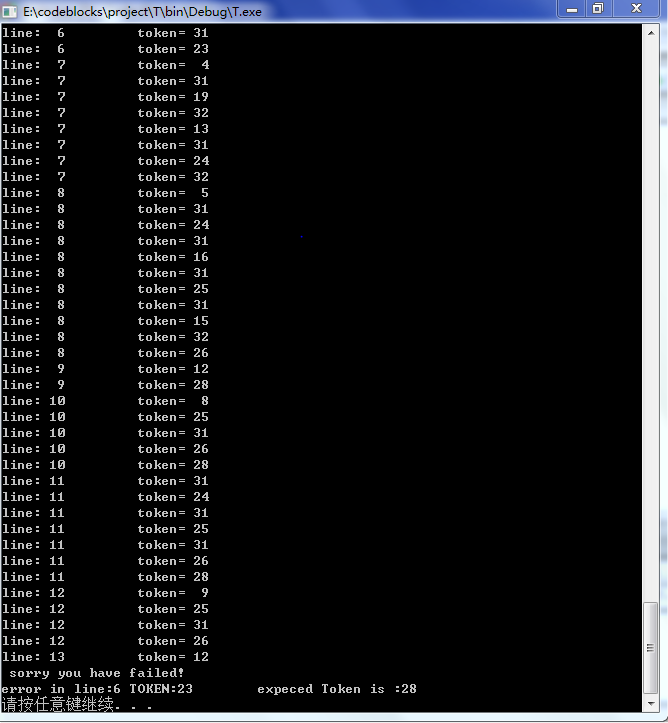
****



运行成功截图：



运行失败截图：





**九、实验结论：**

1检查所处理语言的文法是否满足递归下降算法要求，如不满足，对其进行改造；

2根据递归下降算法编写所处理语言的递归下降程序；

3调用词法分析器，获取测试程序的词法记号；

4将词法分析后的符号表输出到文件symtab1.txt；

5对测试程序的词法记号进行分析、检查，输出分析结果（接受或拒绝），并将语法分析所得的错误信息输出到文件error.txt；

6将语法分析后的符号表输出到文件symtab2.txt；

7选做部分将详细的分析步骤，包括每一步使用的产生式、已分析过的串、当前分析字符、剩余串输出到detail.txt

**十、总结及心得体会：**

由于本实验采用的是递归调用，所以难免会产生回溯，所以难免会有部分产生先识别部分代码，然后再发现后面不合适，前面的部分识别的便失效，此时已经记录了已经识别一次，所以在detail.txt文件和symbol2.txt文件输出。

在error.txt文件记录了每次回溯的记录和最深入记录，所以最后一条记录有效。

**十一、对本实验过程及方法、手段的改进建议：**

本实验的开销很大，无法进行错误的自动修复，遇到一次错误便退出，无法连续识别，而且代码写了很多，这些方面还有待改进。

**报告评分：**

**指导教师签字：**