# 递归下降语法分析设计原理与实现

## 程序功能描述

输入词法分析的输出二元组序列，给出输入串是否为该文法定义的算术表达式的判断结果。同时能发现简单的语法错误。

## 数据结构描述

程序包含一个词法分析的二元组结构体current，用于接收二元组序列的信息，结构定义如下，

typedef struct tuple {

int id;//单词编号

string word;//单词内容(仅当单词为标识符时有内容)

}WORD;

current初始化为{ -1,"" }，表示为空。

## 程序结构描述

## 3.1设计方法

递归下降法是对文法的每一非终结符号，都根据相应产生式各候选式的结构，为其编写一个子程序来识别非终结符号所表示的语法范畴。本实验的文法如下，

G[S]: S→V = E

E→TE′

E′→ATE′ | ε

T→FT′

T′→MFT′ | ε

F→(E) | i

A→ + | -

M→ \* |/

V→i

下面是各个非终结符号的递归下降子程序的流程图，

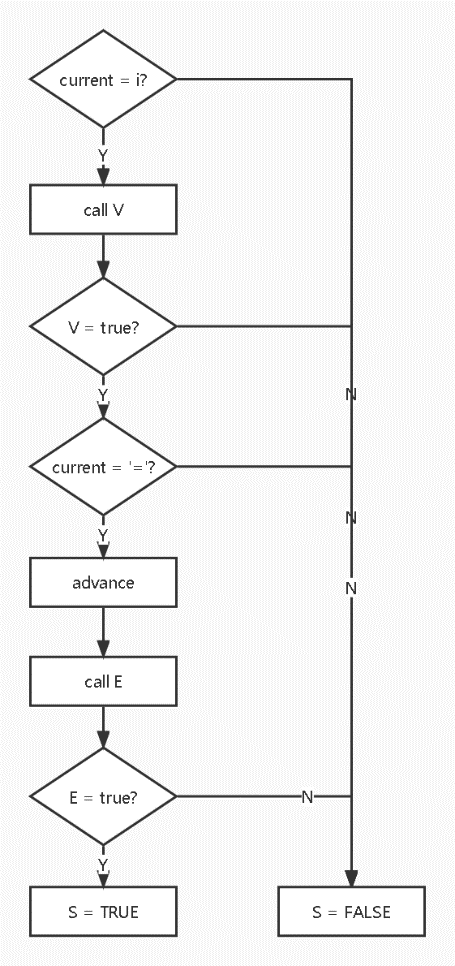


图1 S→V = E

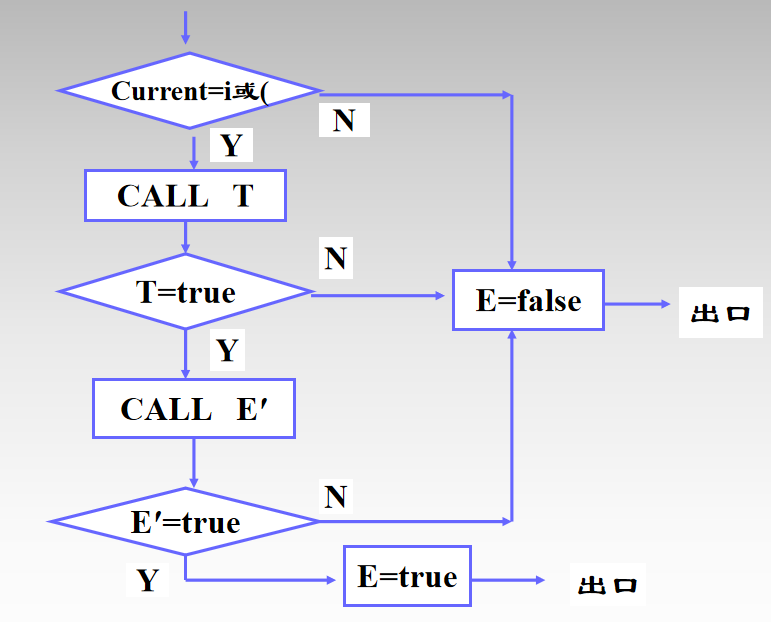


图2 E→TE′

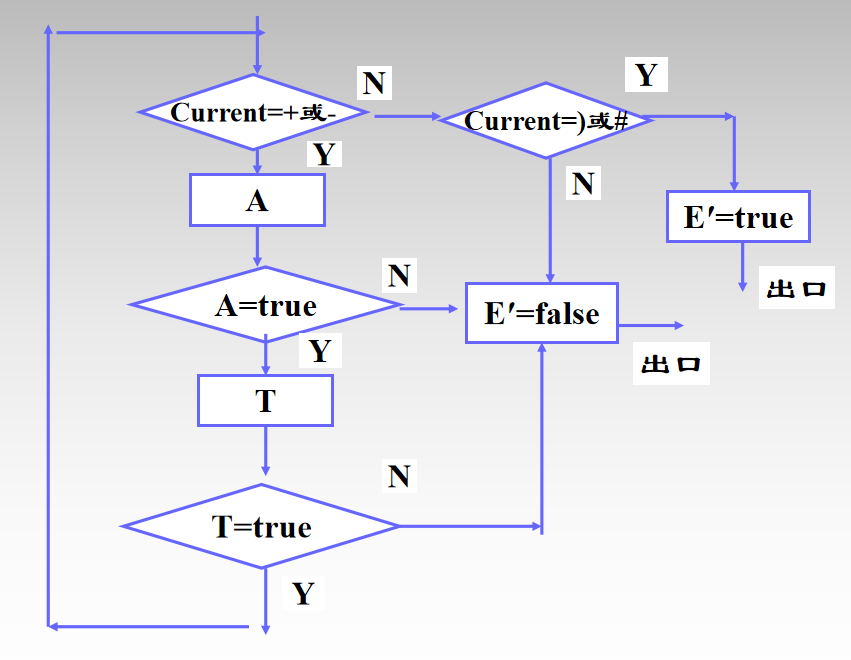


图3 E′→ATE′ | ε

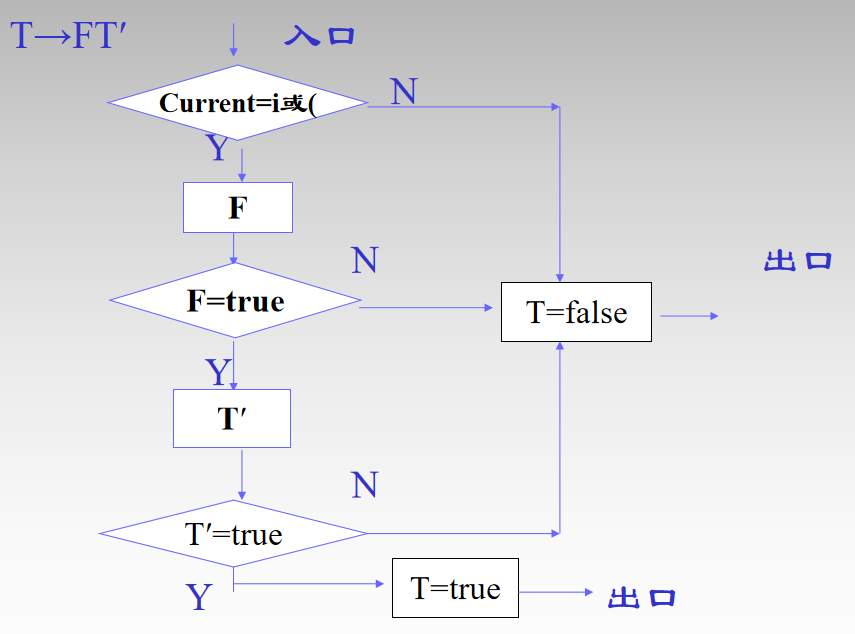


图4 T→FT′

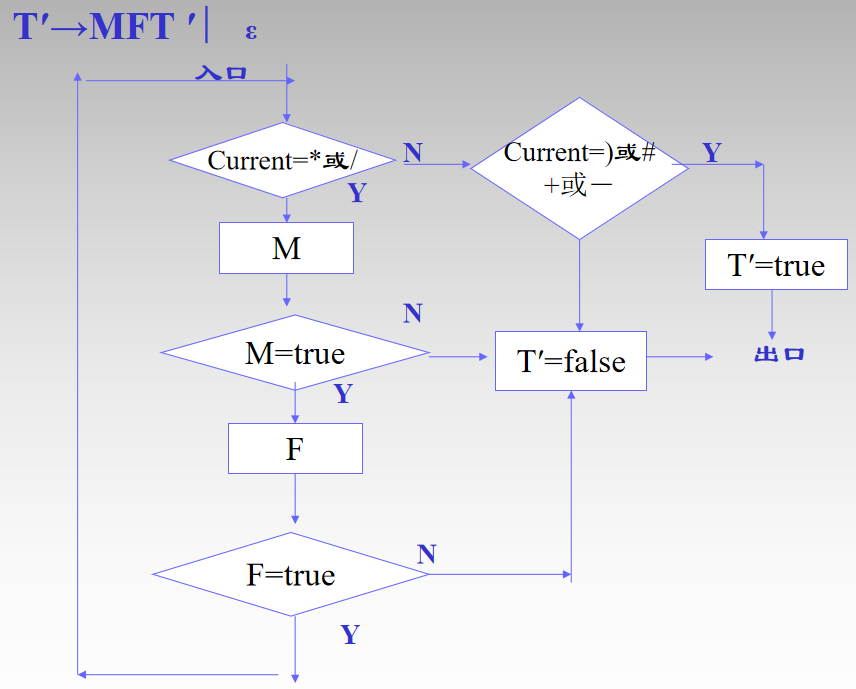


图5 T′→MFT′ | ε

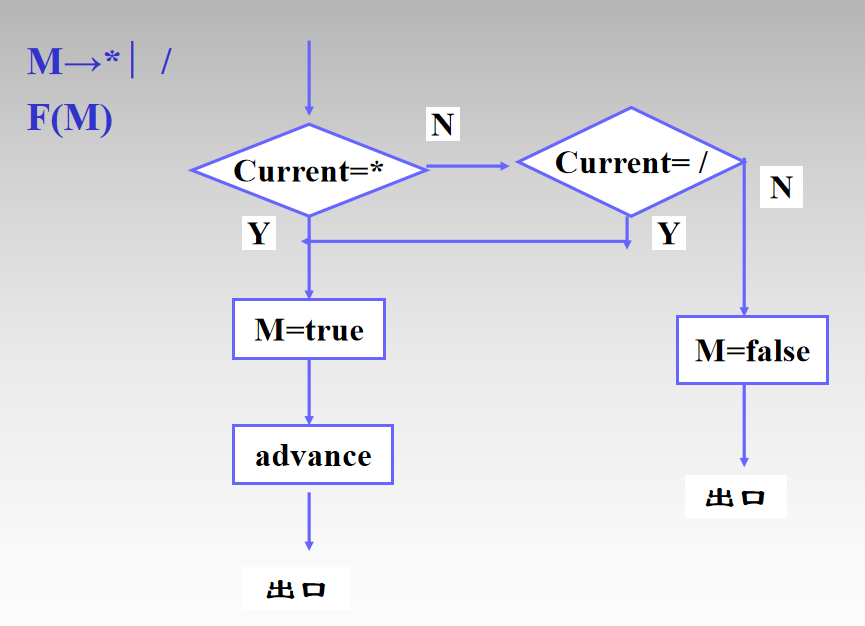


图6 M→ \* |/

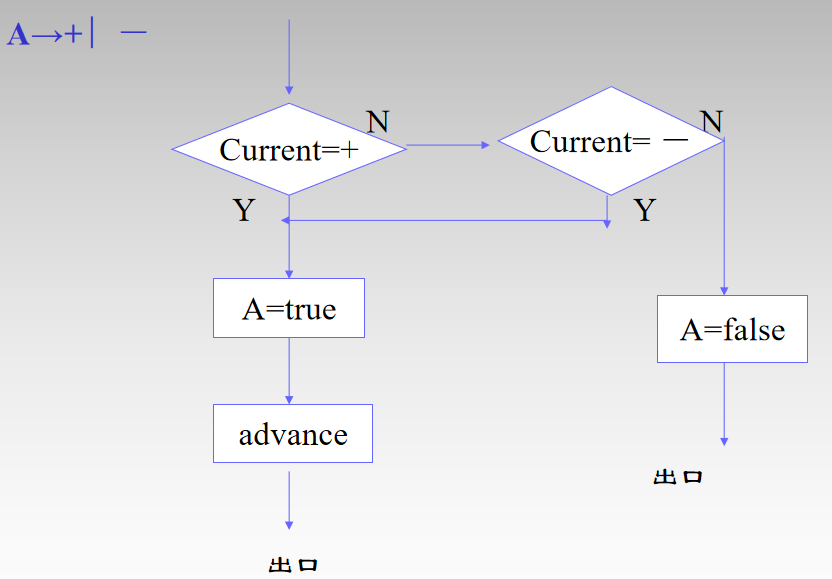


图7 A→ + | -

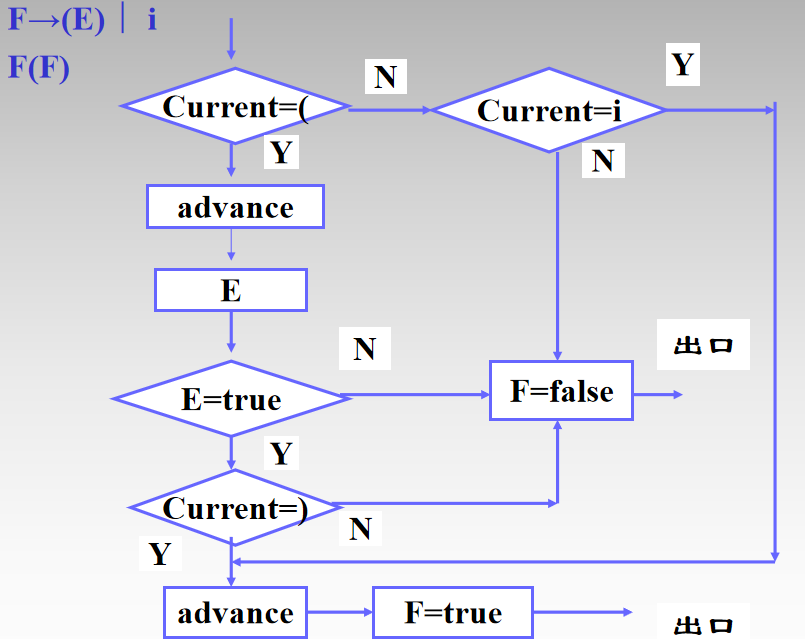


图8 F→(E) | i

## 3.2程序模块

程序分为lex.h、recursive\_descent.h、recursive\_descent.cpp和main.cpp4个部分。其中，lex.h中定义了一系列宏，这些宏的含义是C语言的保留字、标识符和各种其他符号的助记符，用于词法结果的输出。recursive\_descent.cpp负责递归下降语法分析，包括各个非终结符号的递归下降子程序，main.cpp负责将词法分析输出的中间文件传入recursive\_descent.cpp中进行语法检查，若符合语法则在控制台输出符合规则，否则提示出错。

## 3.3函数说明

recursive\_descent.cpp中包含getCurrent、recursive\_descent和各个非终结符号的递归下降子程序。其中getCurrent函数负责获取当前的二元组，将其送往current，输入参数是中间文件，输出参数是二元组; recursive\_descent函数是递归下降的总入口，在其中调用S函数，若成功返回true则符合语法规则，否则不符合语法规则。S函数即为非终结符号S对应的递归下降子程序，类似的还有E、M、V、A等。

这里以A函数为例，其余函数见附录或源代码文件，函数如下所示，

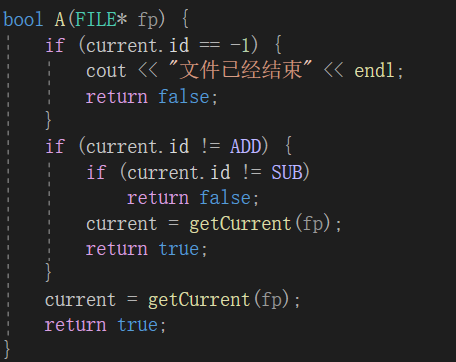


图9 A函数

首先判断当前二元组的id是否为-1，若为-1表示文件结束，则直接退出。否则，根据A的程序框图，判断current是否为+，若是则取下一个二元组到current中，同时返回true。否则判断current是否为-，若不是则返回false，否则返回true。注意这里的getCurrent函数只有在遇到终结符号才能使用。getCurrent函数如下所示，

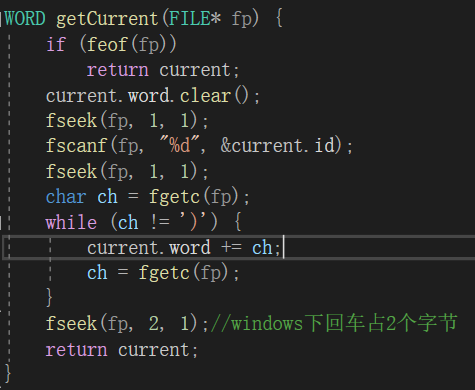


图10 getCurrent函数

还有一点需要说明，文法中的T′在函数中使用TT表示，E′用EE表示。

## 程序测试

第一个测试样例为i=(i+i)\*i#，中间文件recursive\_input.txt如下，



图11 词法分析输出文件

输出结果如下，

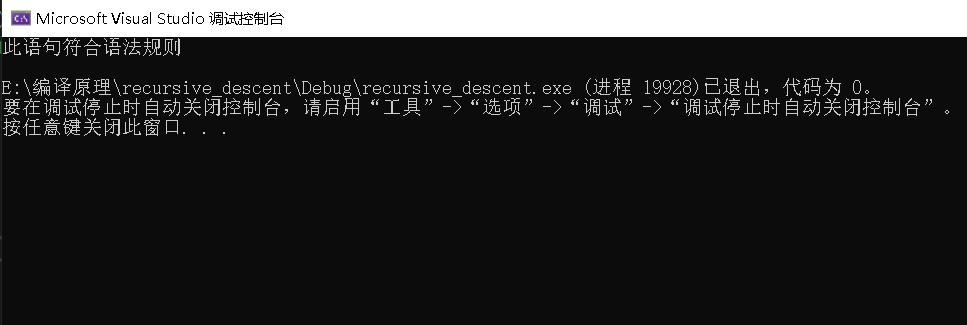


图12 符合语法规则

取一个不符合语法规则的样例i = (i+i)i\*i#，输出结果如下，

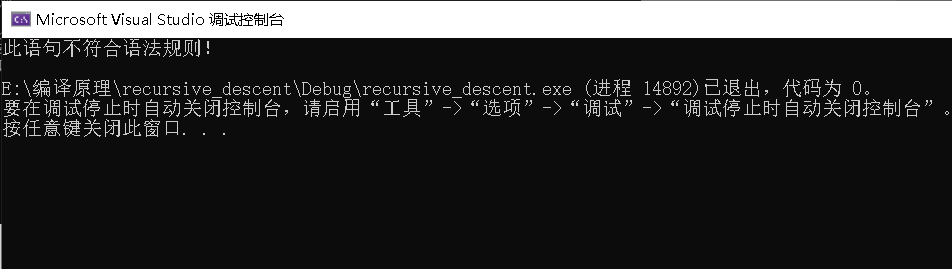


图13 不符合语法规则

# 源代码附录

这里给出recursive\_descent.cpp的内容，

#include "lex.h"

#include "recursive\_descent.h"

typedef struct tuple {

int id;

string word;

}WORD;//词法分析的二元组结构体

WORD current = { -1,"" };

bool E(FILE\* fp);

bool EE(FILE\* fp);

bool T(FILE\* fp);

bool TT(FILE\* fp);

bool F(FILE\* fp);

bool A(FILE\* fp);

bool M(FILE\* fp);

bool V(FILE\* fp);

WORD getCurrent(FILE\* fp) {

if (feof(fp))

return current;

current.word.clear();

fseek(fp, 1, 1);

fscanf(fp, "%d", &current.id);

fseek(fp, 1, 1);

char ch = fgetc(fp);

while (ch != ')') {

current.word += ch;

ch = fgetc(fp);

}

fseek(fp, 2, 1);//windows下回车占2个字节

return current;

}

bool S(FILE\* fp) {

if (!V(fp))

return false;

current = getCurrent(fp);

if (current.id == -1) {

cout << "文件已经结束" << endl;

return false;

}

if (current.id != ASSIGN) {

cout << "缺少=, 不符语法规则" << endl;

return false;

}

current = getCurrent(fp);

if (current.id == -1) {

cout << "文件已经结束" << endl;

return false;

}

if (!E(fp))

return false;

return true;

}

bool E(FILE\* fp) {

if (current.id == -1) {

cout << "文件已经结束" << endl;

return false;

}

if (current.word != "i" && current.id != LEFT\_BRACKET)

return false;

if (!T(fp))

return false;

if (!EE(fp))

return false;

return true;

}

bool EE(FILE\* fp){//E'

if (current.id == -1) {

cout << "文件已经结束" << endl;

return false;

}

if (current.id != ADD && current.id != SUB) {

if (current.id == RIGHT\_BRACKET || current.id == EOF)

return true;

else return false;

}

if (!A(fp))

return false;

if (!T(fp))

return false;

if (EE(fp))

return true;

else return false;

}

bool T(FILE\* fp) {

if (current.id == -1) {

cout << "文件已经结束" << endl;

return false;

}

if (current.word != "i" && current.id != LEFT\_BRACKET)

return false;

if (!F(fp))

return false;

if (!TT(fp))

return false;

return true;

}

bool TT(FILE\* fp) {

if (current.id == -1) {

cout << "文件已经结束" << endl;

return false;

}

if (current.id != MUL && current.id != DIV) {

if (current.id == RIGHT\_BRACKET || current.id == EOF ||

current.id==ADD || current.id==SUB)

return true;

return false;

}

if (!M(fp))

return false;

if (!F(fp))

return false;

if (TT(fp))

return true;

else return false;

}

bool F(FILE\* fp) {

if (current.id == -1) {

cout << "文件已经结束" << endl;

return false;

}

if (current.id != LEFT\_BRACKET) {

if (current.word == "i") {

current = getCurrent(fp);

return true;

}

else {

cout << "含有非法字符！" << endl;

return false;

}

}

current = getCurrent(fp);

if (current.id == -1) {

cout << "文件已经结束" << endl;

return false;

}

if (!E(fp))

return false;

if (current.id == RIGHT\_BRACKET) {

current = getCurrent(fp);

return true;

}

else {

cout << "左右括号不匹配!" << endl;

return false;

}

}

bool A(FILE\* fp) {

if (current.id == -1) {

cout << "文件已经结束" << endl;

return false;

}

if (current.id != ADD) {

if (current.id != SUB)

return false;

current = getCurrent(fp);

return true;

}

current = getCurrent(fp);

return true;

}

bool M(FILE\* fp) {

if (current.id == -1) {

cout << "文件已经结束" << endl;

return false;

}

if (current.id != MUL && current.id != DIV)

return false;

current = getCurrent(fp);

return true;

}

bool V(FILE\* fp) {

current = getCurrent(fp);

if (current.id == -1) {

cout << "文件已经结束" << endl;

return false;

}

if (current.id == 0 && current.word == "i")

return true;

else {

cout << "含有非法字符！" << endl;

return false;

}

}

bool recursive\_descent(FILE\* fp) {

if (S(fp))

return true;

else return false;

}