**编译原理实验报告**

**实验名称：LL（1）**

**2208010421-王旭**

# 实验目的和要求：

编写一个程序，能够识别由词法分析给出的单词符号序列是否是给定文法的正确句子，并输出对输入符号串的分析过程。

# 实验内容：

本实验设计了一个LL(1)语法分析器，主要包括以下功能：

构造预测分析表。

读取输入字符串并进行语法分析。

输出分析过程，包括分析栈、剩余输入串和所用产生式。

# 实验步骤：

****构造程序所需的模块****：

主函数：负责初始化分析表、读取输入字符串并调用分析过程。

输出分析栈函数：打印当前分析栈的内容。

输出剩余串函数：打印当前剩余的输入字符串。

预测分析表：根据给定文法构造预测分析表。

****编写模块代码****：

定义产生式结构体，存储产生式的左部和右部。

初始化预测分析表，填入相应的产生式。

实现主函数，完成输入字符串的读取和语法分析。

****在主函数中调用模块****：

初始化分析栈和剩余输入串。

逐字符读取输入字符串，根据预测分析表进行语法分析。

输出每一步的分析过程。

# 实验代码：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_STACK\_SIZE 20

#define MAX\_PRODUCTIONS 10

#define MAX\_PRODUCTION\_LENGTH 10

char A[MAX\_STACK\_SIZE]; /\* 分析栈 \*/

char B[MAX\_STACK\_SIZE]; /\* 剩余串 \*/

char v1[MAX\_STACK\_SIZE] = {'i', '+', '\*', '(', ')', '#'}; /\* 终结符 \*/

char v2[MAX\_STACK\_SIZE] = {'E', 'G', 'T', 'S', 'F'}; /\* 非终结符 \*/

int j = 0, b = 0, top = 0, l; /\* l为输入串长度 \*/

typedef struct {

char origin; /\* 大写字符 \*/

char array[MAX\_PRODUCTION\_LENGTH]; /\* 产生式右边字符 \*/

int length; /\* 字符个数 \*/

} type;

type e, t, g, g1, s, s1, f, f1; /\* 结构体变量 \*/

type C[MAX\_PRODUCTIONS][MAX\_PRODUCTIONS]; /\* 预测分析表 \*/

void print() { /\* 输出分析栈 \*/

int a;

for (a = 0; a <= top; a++)

printf("%c", A[a]);

printf("\t\t");

}

void print1() { /\* 输出剩余串 \*/

int j;

for (j = 0; j < b; j++) /\* 输出对齐符 \*/

printf(" ");

for (j = b; j <= l; j++)

printf("%c", B[j]);

printf("\t\t\t");

}

int main() {

int m, n, k = 0, flag = 0, finish = 0;

char ch, x;

type cha; /\* 用来接受C[m][n] \*/

/\* 把文法产生式赋值结构体 \*/

e.origin = 'E';

strcpy(e.array, "TG");

e.length = 2;

t.origin = 'T';

strcpy(t.array, "FS");

t.length = 2;

g.origin = 'G';

strcpy(g.array, "+TG");

g.length = 3;

g1.origin = 'G';

g1.array[0] = '^';

g1.length = 1;

s.origin = 'S';

strcpy(s.array, "\*FS");

s.length = 3;

s1.origin = 'S';

s1.array[0] = '^';

s1.length = 1;

f.origin = 'F';

strcpy(f.array, "(E)");

f.length = 3;

f1.origin = 'F';

f1.array[0] = 'i';

f1.length = 1;

/\* 初始化分析表 \*/

for (m = 0; m < 5; m++) {

for (n = 0; n < 6; n++) {

C[m][n].origin = 'N'; /\* 全部赋为空 \*/

}

}

/\* 填充分析表 \*/

C[0][0] = e; C[0][3] = e;

C[1][1] = g; C[1][4] = g1; C[1][5] = g1;

C[2][0] = t; C[2][3] = t;

C[3][1] = s1; C[3][2] = s; C[3][4] = s1; C[3][5] = s1;

C[4][0] = f1; C[4][3] = f;

printf("提示: 本程序只能对由 'i', '+', '\*', '(', ')' 构成的以 '#' 结束的字符串进行分析。\n");

printf("请输入要分析的字符串: ");

do { /\* 读入分析串 \*/

scanf("%c", &ch);

if ((ch != 'i') && (ch != '+') && (ch != '\*') && (ch != '(') && (ch != ')') && (ch != '#')) {

printf("输入串中有非法字符\n");

exit(1);

}

B[j++] = ch;

} while (ch != '#');

l = j; /\* 分析串长度 \*/

ch = B[0]; /\* 当前分析字符 \*/

A[top] = '#'; A[++top] = 'E'; /\* '#', 'E' 进栈 \*/

printf("步骤\t\t分析栈\t\t剩余字符\t\t所用产生式\n");

do {

x = A[top--]; /\* x 为当前栈顶字符 \*/

printf("%d", k++);

printf("\t\t");

for (j = 0; j <= 5; j++) { /\* 判断是否为终结符 \*/

if (x == v1[j]) {

flag = 1;

break;

}

}

if (flag == 1) { /\* 如果是终结符 \*/

if (x == '#') {

finish = 1; /\* 结束标记 \*/

printf("acc!\n"); /\* 接受 \*/

exit(0);

}

if (x == ch) {

print();

print1();

printf("%c 匹配\n", ch);

ch = B[++b]; /\* 下一个输入字符 \*/

flag = 0; /\* 恢复标记 \*/

} else { /\* 出错处理 \*/

print();

print1();

printf("%c 出错\n", ch); /\* 输出出错终结符 \*/

exit(1);

}

} else { /\* 非终结符处理 \*/

for (j = 0; j < 5; j++) {

if (x == v2[j]) {

m = j; /\* 行号 \*/

break;

}

}

for (j = 0; j < 6; j++) {

if (ch == v1[j]) {

n = j; /\* 列号 \*/

break;

}

}

cha = C[m][n];

if (cha.origin != 'N') { /\* 判断是否为空 \*/

print();

print1();

printf("%c-", cha.origin); /\* 输出产生式 \*/

for (j = 0; j < cha.length; j++)

printf("%c", cha.array[j]);

printf("\n");

for (j = cha.length - 1; j >= 0; j--) /\* 产生式逆序入栈 \*/

A[++top] = cha.array[j];

if (A[top] == '^') /\* 为空则不进栈 \*/

top--;

} else { /\* 出错处理 \*/

print();

print1();

printf("%c 出错\n", x); /\* 输出出错非终结符 \*/

exit(1);

}

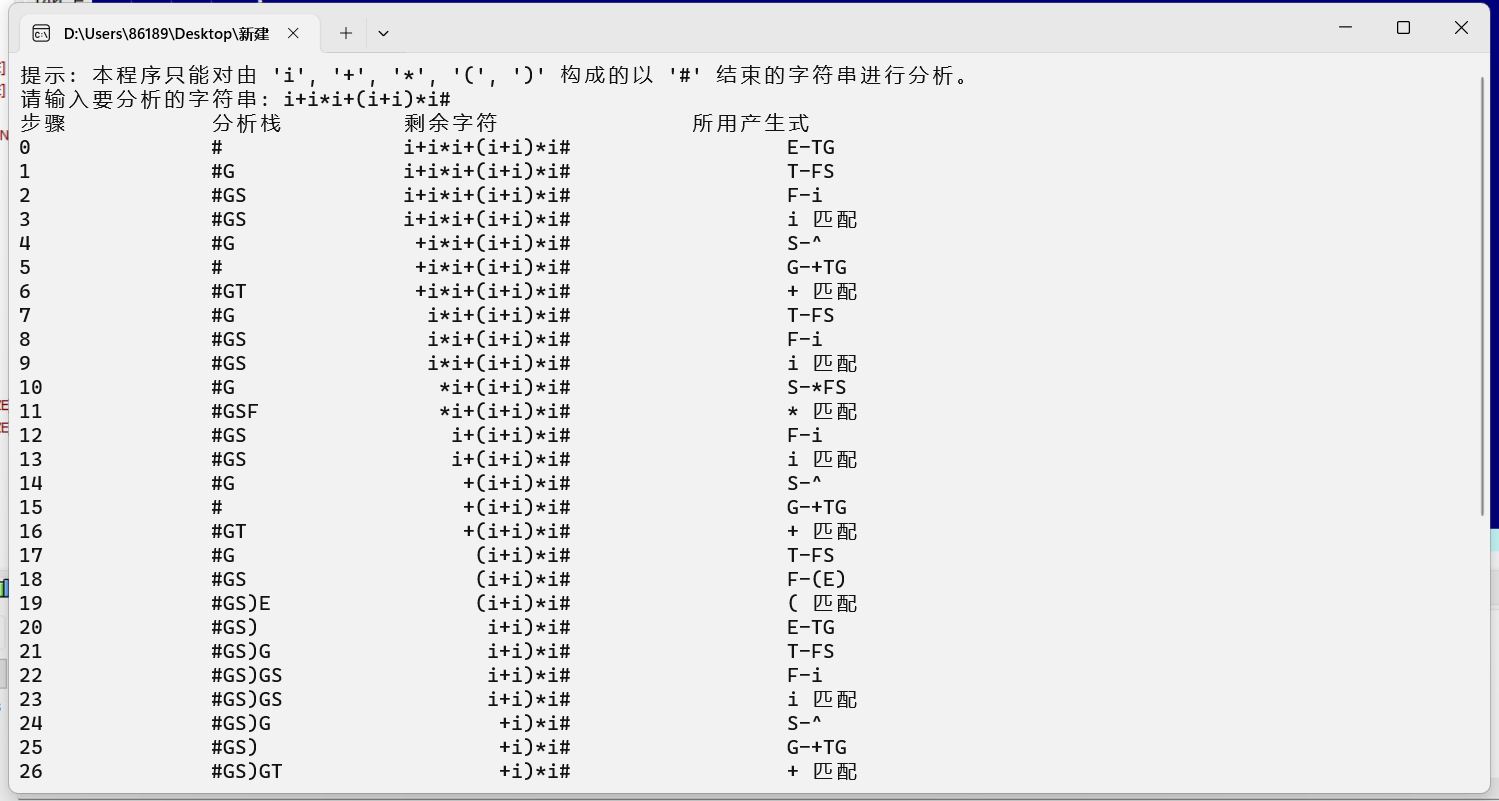
}

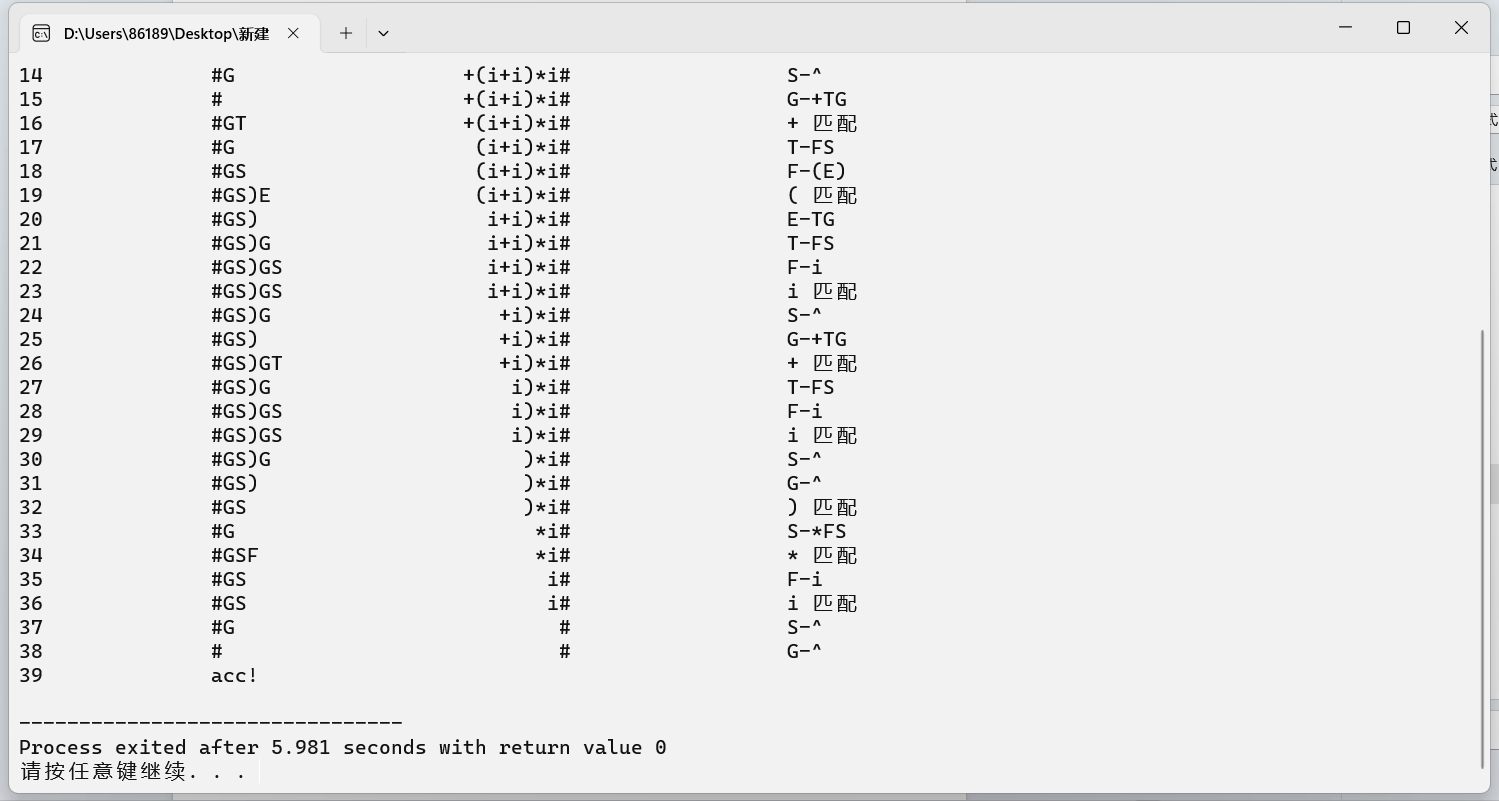
} while (finish == 0);

return 0;

}

# 运行截图：





# 实验总结：

本实验成功实现了LL(1)语法分析器的功能，能够正确识别输入字符串是否符合给定文法，并输出详细的分析过程。通过输入字符串（如 i+i\*i+(i+i)\*i#），程序能够逐步展示分析栈、剩余输入串和所用产生式。