**实验报告：**

**实验名称：LR(1)语法**分析

实验目的和要求

编制一个允许规范族有冲突的项目集用向前查看一个符号的办法来进行处理，并且能够解决存在的无效归约问题，以解决冲突的分析过程。

实验内容和步骤：

**一、实验内容**

对于这个实验，最主要的是构造语法分析表。使用两个二维数组，既action[10][3]和goto[10][2] 用来表示ACTION表和GOTO表。主函数主要完成的功能是能根据当前分析栈中的符号串和向右顺序查看输入串的1个符号就可以唯一确定分析器的动作是移进还是归约和用哪个产生式归约并输出状态栈，符号栈，输入串，ACTION和GOTO。

**二、实验步骤**

1、基于实验的内容，构造程序所需的ACTION表，QOTO表，存放产生式的数组；状态栈，符号栈，输入栈。

2、在主函数中调用各个表和产生式数组来完成所要得到的效果。在本程序中，首先要构造该文法的项目集规范族，并且要包含向前搜索符，接下来根据项目集和转换函数构造LR（1）分析表，包括ACTION表和GOTO表。在主函数中，处理移进，处理归约，输出LR（1）分析表。

实验代码如下：

#include <stdio.h>

#include <string.h>

#include <ctype.h>

/\* LR分析表 \*/

char \*action[10][3] = {

{"S3", "S4", NULL}, /\* 状态0 \*/

{NULL, NULL, "acc"}, /\* 状态1 \*/

{"S6", "S7", NULL}, /\* 状态2 \*/

{"S3", "S4", NULL}, /\* 状态3 \*/

{"r3", "r3", NULL}, /\* 状态4 \*/

{NULL, NULL, "r1"}, /\* 状态5 \*/

{"S6", "S7", NULL}, /\* 状态6 \*/

{NULL, NULL, "r3"}, /\* 状态7 \*/

{"r2", "r2", NULL}, /\* 状态8 \*/

{NULL, NULL, "r2"} /\* 状态9 \*/

};

int goto\_table[10][2] = {

{1, 2}, /\* 状态0 \*/

{0, 0}, /\* 状态1 \*/

{0, 5}, /\* 状态2 \*/

{0, 8}, /\* 状态3 \*/

{0, 0}, /\* 状态4 \*/

{0, 0}, /\* 状态5 \*/

{0, 9}, /\* 状态6 \*/

{0, 0}, /\* 状态7 \*/

{0, 0}, /\* 状态8 \*/

{0, 0} /\* 状态9 \*/

};

/\* 文法符号定义 \*/

char terminals[3] = {'a', 'b', '#'}; /\* 终结符 \*/

char non\_terminals[2] = {'S', 'B'}; /\* 非终结符 \*/

char \*productions[4] = { /\* 产生式集合 \*/

"E->S", /\* 0号产生式 \*/

"S->BB", /\* 1号产生式 \*/

"B->aB", /\* 2号产生式 \*/

"B->b" /\* 3号产生式 \*/

};

/\* 全局变量 \*/

int state\_stack[100]; /\* 状态栈 \*/

char symbol\_stack[100]; /\* 符号栈 \*/

char input\_string[100]; /\* 输入串 \*/

int top\_state = 0; /\* 状态栈顶指针 \*/

int top\_symbol = 0; /\* 符号栈顶指针 \*/

int input\_ptr = 0; /\* 输入串指针 \*/

void print\_stacks() {

/\* 打印状态栈 \*/

printf("状态栈: [");

for (int i = 0; i <= top\_state; i++) {

printf("%d", state\_stack[i]);

if (i < top\_state) printf(", ");

}

printf("]\t");

/\* 打印符号栈 \*/

printf("符号栈: [");

for (int i = 0; i <= top\_symbol; i++) {

printf("%c", symbol\_stack[i]);

if (i < top\_symbol) printf(", ");

}

printf("]\t");

/\* 打印剩余输入串 \*/

printf("输入串: ");

for (int i = input\_ptr; input\_string[i] != '#'; i++) {

printf("%c", input\_string[i]);

}

printf("#\t");

}

int get\_terminal\_index(char c) {

for (int i = 0; i < 3; i++) {

if (terminals[i] == c) return i;

}

return -1; /\* 不是终结符 \*/

}

int get\_non\_terminal\_index(char c) {

for (int i = 0; i < 2; i++) {

if (non\_terminals[i] == c) return i;

}

return -1; /\* 不是非终结符 \*/

}

void lr\_parser() {

int step = 1;

state\_stack[0] = 0; /\* 初始状态0入栈 \*/

symbol\_stack[0] = '#'; /\* 栈底符号 \*/

printf("步骤\t状态栈和符号栈\t\t\t输入串\t\t动作\n");

printf("------------------------------------------------------------\n");

while (1) {

printf("%d\t", step++);

print\_stacks();

char current\_char = input\_string[input\_ptr];

int terminal\_idx = get\_terminal\_index(current\_char);

if (terminal\_idx == -1) {

printf("\n错误: 非法输入字符 '%c'\n", current\_char);

return;

}

char \*action\_str = action[state\_stack[top\_state]][terminal\_idx];

if (action\_str == NULL) {

printf("\n错误: 在状态%d遇到字符'%c'没有定义的动作\n",

state\_stack[top\_state], current\_char);

return;

}

/\* 移进动作 \*/

if (action\_str[0] == 'S') {

int new\_state = action\_str[1] - '0';

printf("移进 %s\n", action\_str);

/\* 状态入栈 \*/

state\_stack[++top\_state] = new\_state;

/\* 符号入栈 \*/

symbol\_stack[++top\_symbol] = current\_char;

/\* 输入指针前移 \*/

input\_ptr++;

}

/\* 归约动作 \*/

else if (action\_str[0] == 'r') {

int prod\_num = action\_str[1] - '0';

char \*production = productions[prod\_num];

printf("归约 %s (%s)\t", action\_str, production);

/\* 获取产生式左部 \*/

char left\_symbol = production[0];

/\* 获取产生式右部长度 \*/

int rhs\_length = strlen(production) - 3;

/\* 弹出状态栈和符号栈中的符号 \*/

top\_state -= rhs\_length;

top\_symbol -= rhs\_length;

/\* 获取新的栈顶状态 \*/

int current\_state = state\_stack[top\_state];

/\* 获取左部非终结符的索引 \*/

int nt\_index = get\_non\_terminal\_index(left\_symbol);

if (nt\_index == -1) {

printf("\n错误: 非终结符 '%c' 未定义\n", left\_symbol);

return;

}

/\* 查找goto表 \*/

int new\_state = goto\_table[current\_state][nt\_index];

if (new\_state == 0) {

printf("\n错误: 在状态%d对'%c'没有GOTO转移\n",

current\_state, left\_symbol);

return;

}

/\* 新状态入栈 \*/

state\_stack[++top\_state] = new\_state;

/\* 左部符号入栈 \*/

symbol\_stack[++top\_symbol] = left\_symbol;

printf("GOTO %d\n", new\_state);

}

/\* 接受动作 \*/

else if (strcmp(action\_str, "acc") == 0) {

printf("接受\n");

printf("\n分析成功! 输入串符合文法规则。\n");

return;

}

}

}

int main() {

printf("实验二: LR分析器的设计\n");

printf("支持的终结符: a, b, #\n");

printf("文法产生式:\n");

for (int i = 0; i < 4; i++) {

printf("%d: %s\n", i, productions[i]);

}

printf("\n");

printf("请输入要分析的字符串(以#结束): ");

char c;

int i = 0;

/\* 读取输入 \*/

do {

scanf("%c", &c);

if (c != ' ' && c != '\n') { /\* 跳过空格和换行 \*/

if (c != 'a' && c != 'b' && c != '#') {

printf("错误: 非法字符 '%c'。只允许输入a, b, #\n", c);

return 1;

}

input\_string[i++] = c;

}

} while (c != '#');

printf("\n开始分析过程:\n");

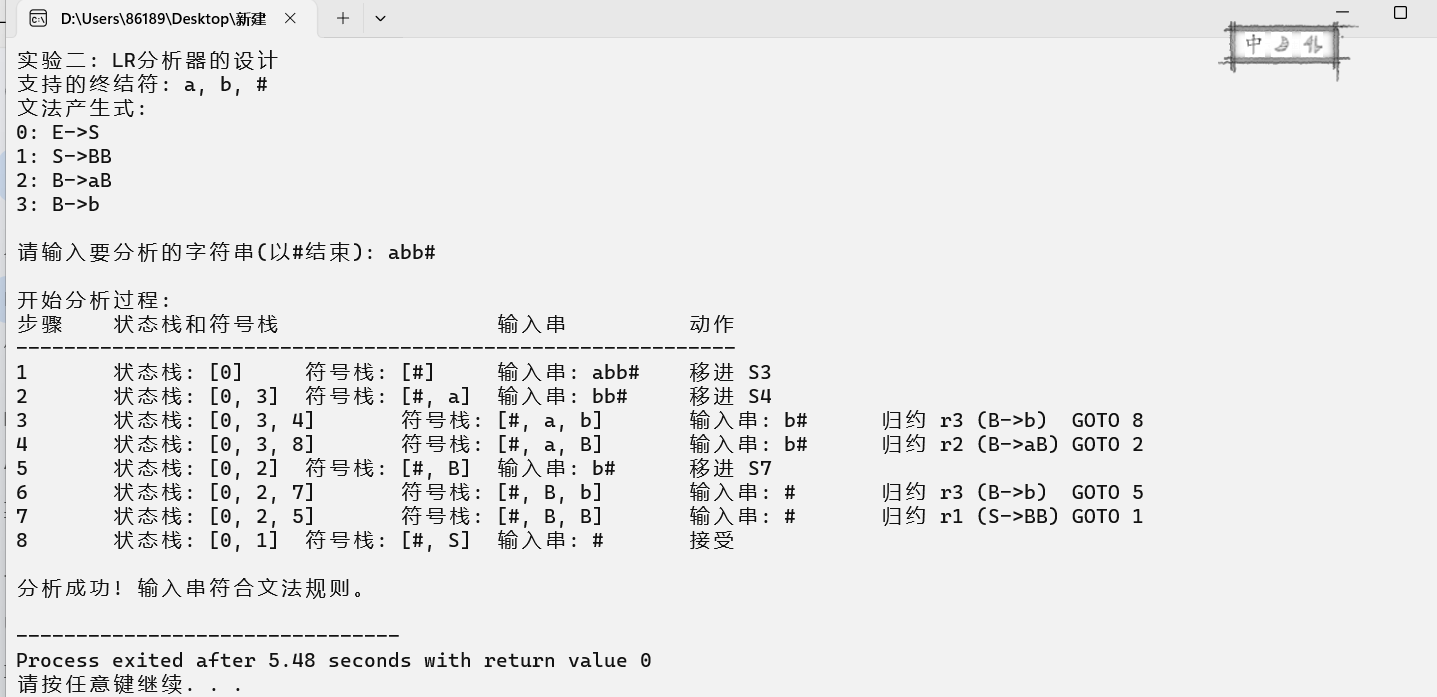
lr\_parser();

return 0;

}

**三、实验过程记录：**

实验截图：



**四、实验总结：**

本次实验成功实现了一个简单的LR分析器，能够对给定的输入串进行语法分析。通过实验，深入理解了LR分析的基本原理，并掌握了LR分析表的构造方法