1）示范,示例 LL（1）文法如下：

G[A]:

(1)S::=pA (2)S::=qB (3)A::=cAd

(4)A::=a (5)B::=dB (6)B::=b

对应每条规则的选择集如下：

(1){p} (2){q} (3){c}

(4){a} (5){d} (6){b}

2）下面是简化的四则运算的语法规则，递归下降子程序法不能直接用这组规则进行语法分析 G[E]:

E::=E+T E::=T T::=T*F

T::=F F::=(E) F::=i

但是我们可以通过改造文法后，构建一等价文法，可以利用递归下降子程序发进行语法分析，该等价文法如下：

1.E::=TA 2.A::=+TA 3.A::=

4.T::=FB 5.B::=*FB 6.B::=

7.F::=(E) 8.F::=i

对应规则的选择集如下：

1.{(,i} 2.{+} 3.{),#} 4.{(,i}

5.{*} 6.{+,),#} 7.{(} 8.{i}

任务：编写递归下降子程序进行语法分析

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SourceMaxLength 10000
//子程序声明
void S();
void A();
void B();
struct sourceStr{
  char source[SourceMaxLength];//存储源码
  int pointer;//指针，控制取字符
};
struct sourceStr sour;
char token;//暂存当前取出的字符
int flag = 1;//编译成功标记
//获取一个字符
char getToken()
{
  token = sour.source[sour.pointer];//取出当前指针指向的字符
  sour.pointer++;//指针后移
  return token;
}
//编译出错
void error()
{
  flag = 0;//将标志位置0,即编译失败
  printf("error;");//打印错误提示信息
}
//子程序S
void S()
{
  token = getToken();//获取一个字符
  if (token != 'p' && token != 'q')
      error();//编译出错
  else
  {
      if (token == 'p')//满足此条件
          A();//调用子程序A
      else
          B();//满足此条件则调用子程序A
  }
}
//子程序A
void A()
{
  token = getToken();//获取一个字符
  if (token != 'c'&&token != 'a')//不满足递归条件
      error();//编译出错
  else{
      if (token == 'c')//满足此条件
      {
          A();//调用子程序A
          token = getToken();//获取字符
          if (token != 'd')//不满足条件
```

```
                error();//编译出错
            }
            else{
                ;
            }
        }
    }
    //子程序B
    void B()
    {
        token = getToken();//获取一个字符
        if (token != 'd'&&token != 'b')//不满足递归条件
            error();//编译出错
        else{
            if (token == 'd')//满足此条件
            {
                B();//调用子程序B
            }
        }
    }
    int main()
    {
        sour.pointer = 0;//指针指向第一个位置
        strcpy(sour.source, "pcaadd");//将字符串复制到源程序变量
        S();//开始子程序调用
        if (flag == 1)//编译成功
        {
            printf("success!");//打印成功提示信息
        }
        else//编译失败
        {
            printf("error");//打印失败提示信息
        }
        return 0;
    }
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SourceMaxLength 10000
//子程序声明
void A();
void B();
void E();
void T();
void F();
struct sourceStr{
    char source[SourceMaxLength];//存放源代码
    int pointer;//指针，控制取字符
};
struct sourceStr sour;
char token;//暂存当前取出的字符
int flag = 1;//编译成功标记
//获取一个字符
char getToken()
{
    token = sour.source[sour.pointer];//取出当前指针指向的字符
    sour.pointer++;//指针后移
    return token;
}
//编译出错
void error()
{
    flag = 0;//将标志位置0,即编译失败
    printf("error;");
}
//子程序E
void E()
{
    T();//调用子程序T
    A();//调用子程序A
}
//子程序T
void T()
{
    F();//调用子程序F
    B();//调用子程序B
}
//子程序A
void A()
{
    token = getToken();//获取一个字符
```

```
    if (token != '+'&&token != ')'&&token != '#')//不满足递归条件
        error();//编译出错
    else{
        if (token == '+')//满足此条件
        {
            T();//调用子程序T
            A();//调用子程序A
        }
        else{//满足此条件
            sour.pointer--;//为空，指针前移，继续其他子程序递归判断
        }
    }
}
//子程序B
void B()
{
    token = getToken();//获取一个字符
    if (token != '*'&&token != '+'&&token != ')'&&token != '#')//不满足递归条件
        error();//编译出错
    else{
        if (token == '*')//满足此条件
        {
            F();//调用子程序F
            B();//调用子程序B
        }
        else{//满足此条件
            sour.pointer--;//为空，指针前移，继续其他子程序递归判断
        }
    }
}
//子程序F
void F()
{
    token = getToken();//获取一个字符
    if (token != '('&&token != 'i')//不满足递归条件
    {
        error();//编译出错
    }
    else
    {
        if (token == '(')//满足此条件
        {
            E();//调用子程序E
            if (token != ')')//不满此条件
                error();//编译出错
        }
        else {//满足此条件
            ;
        }
    }
}
int main()
{
    char t[SourceMaxLength];//存储输入的表达式
    while (gets(t))//以便多次输入测试数据
    {
        sour.pointer = 0;//指针指向第一个位置
        strcpy(sour.source, t);//将字符串复制到源程序变量
        E();//开始子程序调用
        if (flag == 1)//编译成功
        {
            printf("success!\n");
        }
        else//编译失败
        {
            printf("error!\n");
        }
    }
    return 0;
}
```