**编译原理**

班级：20计算机科学与计算机

姓名：陈伟剑 学号：2020333503081

实验二 递归下降语法分析器的构建

1. 实验要求

运用递归下降法，针对给定的上下文无关文法，给出实验方案。预估实验中可能出现的问题。

1. 实验方案

（评价依据实验方案设计是否合理，包括输入输出的设计）

①消除左递归

*lexp->atom|list*

*atom->number|identifier*

*list->(lexp-seq)*

*lexp-seq->lexp lexp-seq’*

*lexp-seq’->lexp lexp-seq’|ε*

②根据LL(1),对每一个文法设计一个函数

③需要的函数

int lexp(); *// lexp->atom|list*

void atom(); *// atom->number|identifer*

void list(); *// list->(lexp-seq)*

void lexp\_seq(); *// lexp-seq->lexp lexp-seq’*

void lexp\_seq\_(); *// lexp-seq’->lexp lexp-seq’|ε*

1. 预估问题

（是否有预估的问题，预估的问题是否合理）

要确保转换后的文法符合LL(1)的定义，否则程序会出错。

理论基础（评价依据 理论知识非常清楚）

LL(1)满足的条件：

①文法不含左递归

②对于每个非终结符A的各个候选是的终结首符号集两两不相交。即，若A🡪a1|a2|…|an，则FIRST(ai)∩FIRST(aj)=Φ，其中1<=i , j<=n，且i≠j

③对于文法中每个非终结符A，若它的某个候选是的终结首符号集包含ε，则FIRST(a)∩FOLLOW(A)=Φ

递归下降分析程序的思想是：识别程序由一组子程序组成。每个子程序对应于一个非终结符。

所谓推导，在程序种体现为每个文法函数按情况互相调用。比如文法lexp🡪atom|list,先调用函数lexp(),经判断后，要推导出list，于是调用函数list()，根据文法list🡪(lexp-seq)，先匹配终结符“(”，再递归调用函数lexp-seq。以此结合文法和输入串反复进行函数递归调用。

1. 内容和步骤

1、针对4.8习题输入和输出的设计及代码

2、针对现场给定语法的设计和处理

【考虑简单算术表达式文法G:

E→E + T | T

T→T \* F | F

F→(E) | id

试设计递归下降分析程序，以对任意输入的符号串进行语法分析。】

消除左递归：

E->TE’

E’->+TE’|ε

T->FT’

T’->\*FT’|ε

F->(E)|id

构造FIRST集合：

First(E)={(,id}

First(E’)={+, ε}

First(T)={(,id}

First(T’)={, ε}

First(F)={(,id}

构造FOLLOW集合：

Follow(E)={KaTeX parse error: Expected 'EOF',

got '}' at position 3: ,)}̲ Follow(E’)={,)}

Follow(T)={+,KaTeX parse error: Expected 'EOF',

got '}' at position 3: ,)}̲ Follow(T’)={+,,)}

Follow(F)={\*,+,$,)}

3、实验具体步骤

①输入待测字符串，回车以完成输入

②按顺序匹配字符，若符合则继续匹配直到匹配完成，否则报错结束程序

1. 实验结果:

5.1针对4.8习题输入和输出的设计及代码

①代码

**#include**<iostream>

**#include**<string.h>

**using** **namespace** std;

// grammar 已消除左递归

/\*

    lexp->atom|list

    atom->number|identifier

    list->(lexp-seq)

    lexp-seq->lexp lexp-seq’

    lexp-seq’->lexp lexp-seq’|ε

\*/

//=========global virables===========

**int** index**=**0;  //记录当前扫描的位置

**int** level**=**0;  //记录当前语法树的深度

string str;   //存储输入的串

//=========functions==========

**int** lexp();     // lexp->atom|list

**void** atom();    // atom->number|identifer

**void** list();    // list->(lexp-seq)

**void** lexp\_seq();    // lexp-seq->lexp lexp-seq’

**void** lexp\_seq\_();   // lexp-seq’->lexp lexp-seq’|ε

**void** printLevel(){ //根据当前的level确定当前递归树结点的深度

    cout**<<**"str["**<<**index**<<**"]="**<<**str**[**index**]<<**"  ";

**for**(**int** i**=**0;i**<**level;i**++**){

        cout**<<**"----";  //每加深一层，前缀就多一个"----"

    }

}

// lexp->atom|list

**int** lexp(){

**if**( (str**[**index**]>=**'0'**&&**str**[**index**]<=**'9')**||**(str**[**index**]>=**'a'**&&**str**[**index**]<=**'z') ){

        level**++**; atom(); level**--**;

**return** 1;

    }

**else** **if**(str**[**index**]==**'('){

        level**++**; list(); level**--**;

**return** 1;

    }

**else** {

**return** 0;

    }

}

// atom->number|identifer

**void** atom(){

**if**(str**[**index**]>=**'0'**&&**str**[**index**]<=**'9'){

        printLevel();

        cout**<<**str**[**index**]<<**endl;

        index**++**;

    }

**else** **if**(str**[**index**]>=**'a'**&&**str**[**index**]<=**'z'){

        printLevel();

        cout**<<**str**[**index**]<<**endl;

        index**++**;

    }

}

// list->(lexp-seq)

**void** list(){

    //如果符合文法，一开始匹配的符号一定是')'

**if**(str**[**index**]!=**'('){

        exit(**-**1);

    }

    // '('终结符完成匹配后，index指向下一个预读的符号

    printLevel();

    cout**<<**str**[**index**]<<**endl;

    index**++**;

    level**++**; lexp\_seq(); level**--**;

    //lexp-seq推导完后，如果符合文法，那么index现在指向的字符一定是`)`

**if**(str**[**index**]!=**')'){

        printLevel();

        cout**<<**"list()error: ')' is excepted "**<<**endl;

        exit(**-**1);

    }

    printLevel();

    cout**<<**str**[**index**]<<**endl;

    index**++**;

}

// lexp-seq->lexp lexp-seq’

**void** lexp\_seq(){

    level**++**; lexp(); level**--**;

    level**++**; lexp\_seq\_(); level**--**;

}

// lexp-seq’->lexp lexp-seq’|ε

**void** lexp\_seq\_(){

    // 如果预读到的符号可以完成lexp的推导，说明本次推导选择的是第一项

**int** temp**=**lexp();

**if**(temp**==**1){

        level**++**; lexp\_seq\_(); level**--**;

    }

**else** {

**return** ;

    }

}

**int** main(**int** argc,**char\*** argv[]){

    cin**>>**str;

**int** len**=**str.size();

    str**[**len**++]=**'$';

    //可能有多个外括号，如(a)(b)

**while**(index**<**len**-**1){lexp();}

    //如果能顺利读取到结束符号，则说明输入的串符合文法

**if**(str**[**index**]==**'$') cout**<<**"accept."**<<**endl;

**else** cout**<<**"wrong answer."**<<**endl;

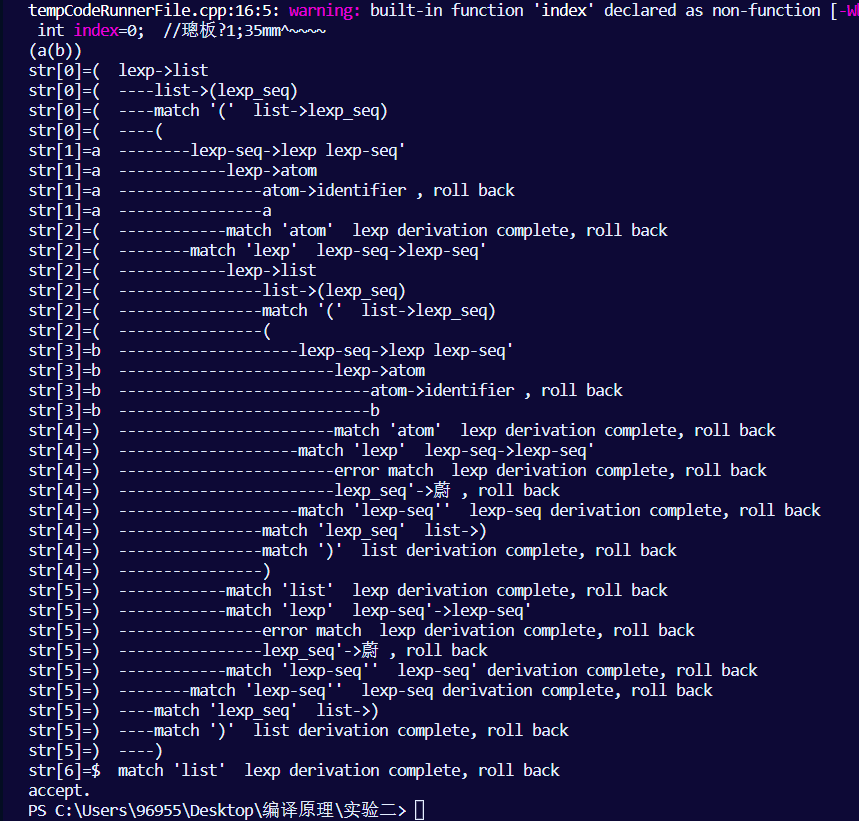
**return** 0;

}

②截图



②附带路径记录的截图



5.2简单算术表达式文法G

①代码

**#include**<iostream>

**#include**<string.h>

**using** **namespace** std;

// grammar 已消除左递归

/\*

    E->TE’

    E’->+TE’|ε

    T->FT’

    T’->\*FT’|ε

    F->(E)|id

\*/

//=========global virables===========

**int** index**=**0;  //记录当前扫描的位置

**int** level**=**0;  //记录当前语法树的深度

string str;   //存储输入的串

//=========functions==========

**void** E();       // E->TE’

**void** E\_();  // E’->+TE’|ε

**void** T();   // T->FT’

**void** T\_();  // T’->\*FT’|ε

**int** F();    // F->(E)|id

**void** printLevel(){ //根据当前的level确定当前递归树结点的深度

    cout**<<**"str["**<<**index**<<**"]="**<<**str**[**index**]<<**"  ";

**for**(**int** i**=**0;i**<**level;i**++**){

        cout**<<**"----";  //每加深一层，前缀就多一个"----"

    }

}

// E->TE’

**void** E(){

    level**++**; T(); level**--**;

    level**++**; E\_(); level**--**;

}

// E’->+TE’|ε

**void** E\_(){

    //如果一开始匹配的符号是'+',则匹配第一项，否则匹配空

**if**(str**[**index**]!=**'+'){

**return** ;

    }

**else**{

        printLevel();

        cout**<<**str**[**index**]<<**endl;

        index**++**;

        level**++**; T(); level**--**;

        level**++**; E\_(); level**--**;

    }

}

// T->\*FT’

**void** T(){

    level**++**; F(); level**--**;

    level**++**; T\_(); level**--**;

}

// T’->\*FT’|ε

**void** T\_(){

    //如果预读的符号可以完成F的推导，则选择第一项

**if**(str**[**index**]!=**'\*'){

**return** ;

    }

**else**{

        printLevel();

        cout**<<**str**[**index**]<<**endl;

        index**++**;

        level**++**; F(); level**--**;

        level**++**; T\_(); level**--**;

    }

}

// F->(E)|id

**int** F(){

    //如果一开始匹配的符号不是'(',则选择推导id

**if**(str**[**index**]!=**'('){

        printLevel();

        cout**<<**str**[**index**]<<**endl;

        index**++**;

**return** 0;

    }

    // '('终结符完成匹配后，index指向下一个预读的符号

    printLevel();

    cout**<<**str**[**index**]<<**endl;

    index**++**;

    level**++**; E(); level**--**;

    //lexp-seq推导完后，如果符合文法，那么index现在指向的字符一定是`)`

**if**(str**[**index**]!=**')'){

        printLevel();

        cout**<<**"list()error: ')' is excepted "**<<**endl;

        exit(**-**1);

    }

    printLevel();

    cout**<<**str**[**index**]<<**endl;

    index**++**;

**return** 1;

}

**int** main(**int** argc,**char\*** argv[]){

    cin**>>**str;

**int** len**=**str.size();

    str**[**len**++]=**'$';

**while**(index**<**len**-**1){E();;}

**if**(str**[**index**]==**'$') cout**<<**"accept."**<<**endl;

**else** cout**<<**"wrong answer."**<<**endl;

**return** 0;

}

②截图



1. 实验结论:

1 、实验结论

设计、编制、调试递归下降语法分析程序，对输入的符号串进行分析匹配，观察输入符号串是否为给定文法的句子，输出其语法树e。

2、分析和总结

1）对输入设计的结论

根据直(间)接消除左递归的方法，优先修改给定文法，使其满足LL(1)的要求。

2）对输出设计的结论

设置一个全局变量level用来记录当前递归的深度，每次匹配到终结符后，根据深度和终结符打印出期望的格式，通过level的变换可以使递归过程体现的更接近树的形式。

3）对递归下降法的算法的结论

整体思路为：从根节点出发，自顶向下为输入串寻找一个最左匹配序列，建立一棵语法树。文法的每个非终结符对应一个函数。分析过程就是从文法开始符出发执行一组递归过程。

3、对预估问题的结论

未遇到预估问题