

评阅项1计分＿＿＿＿＿＿

评阅项2计分＿＿＿＿＿＿

评阅项3计分＿＿＿＿＿＿

评阅项4计分＿＿＿＿＿＿

实验得分＿＿＿＿＿＿

评阅日期＿＿＿＿＿＿

**编译原理实验**

实验一：自顶向下语法分析

实验选题：自顶向下语法分析

学生姓名：

学 号：

专业班级：

任课教师： 曹江莲

提交日期：

1. **实验目的**

通过本次实验，加深对LL(1)分析法中FIRST与FOLLOW集合的计算以及LL（1）文法的判定算法与分析过程的理解，学习程序设计语言的语法分析器的手工编程方法。

1. **实验内容**

（1）将递归下降子程序分析方法运用于算术表达式文法；

（2）生成如图所示的抽象语法树(Abstract Syntax Tree)，设计并实现一个简单的计算器。

1. **算法思想及问题对策**

1. 算法描述（可以用流程图）

该程序主要用来模拟递归下降分析器的分析过程，写入程序的文法是(^代表特殊记号，$代表空：

G[E]:

E-->TE^

E^-->+TE^|$

T-->FT^

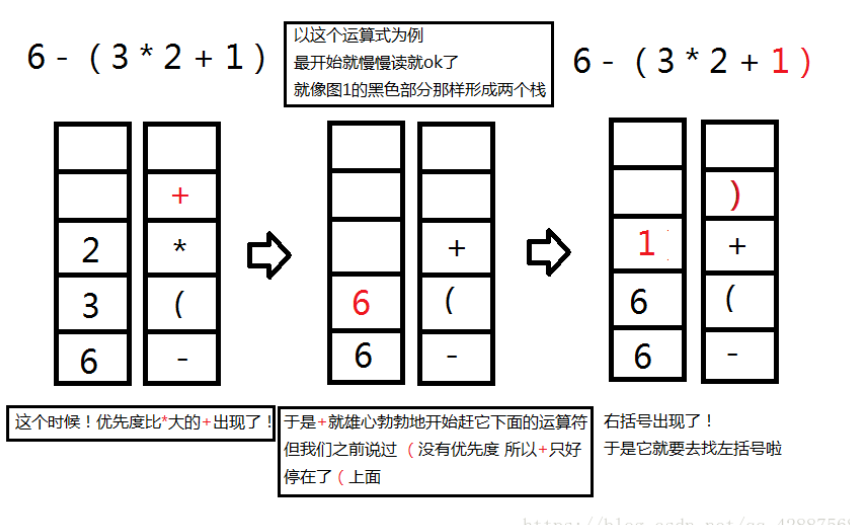
T-->\*FT^|$

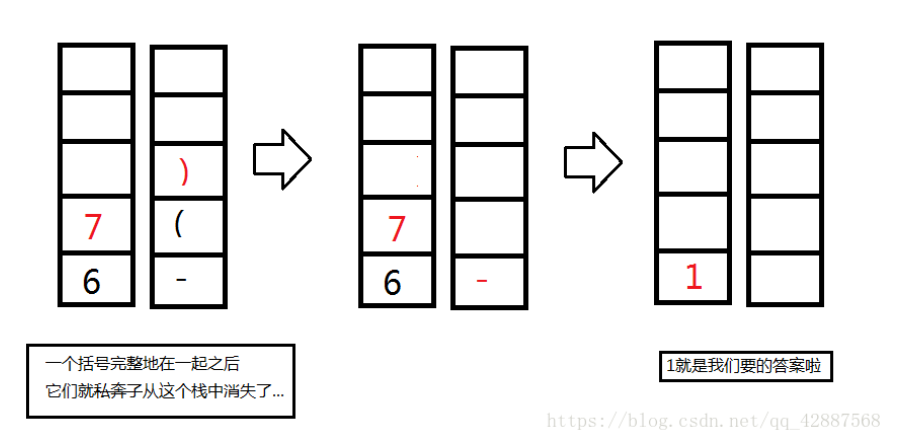
F-->(E)|i

i-->(0|1|2|3|4|5|6|7|8|9)\*

然后模拟递归下降分析，用递归的方法模拟栈，来实现字符串的分析过程。先执行某个函数代表将某个产生式的右部逆序压栈，该函数代表产生式右部的最左边的元素。

计算器的编写较为麻烦，主要一开始没想没那么多，在网上查找资料后，将输入的算术表达式拆分成2个栈，算符栈和数字栈，然后由于不同的算符之间有不同的优先度，我们这里只有+和\*，+的优先度高于\*（注意，不是优先级），当优先度高的符号进入栈中 所有比它优先度低的符号都要弹出，就是出现一个高优先度的符号时，其前面直到比他优先度高的算符出现之间的算符全部合并为一个结果。





2．设计过程中的问题与对策

刚开始我是想可以动态输入CFG然后来进行递归下降分析，然而那个时候的知识储备还不够，浪费了很久的时间也没有搞出来，所以最后和同学讨论得知可以将CFG写死在程序中，所以就采用了这种方法。（其实现在想想输入CFG然后进行判断其实也不难，主要是当时想的太复杂了，而且好像第二个实验更加的直观明了）

对于计算器的实现，我是参考了网上资料的，由于用一个栈进行算术处理要进行string类型到int类型的转换很麻烦，所以就采用了2个栈。

3. 本实验的特色

能够多次执行程序，计算器的精度比较高，对于较为复杂的算术表达式也能准确地得出结果。

1. **具体实现**
2. 主要变量

|  |  |
| --- | --- |
|  | **string str;//接受输入的表达式。**  **int input\_num = 0;//用于将字符串转化为int类型值时的中间变量**  **int location = 0;//字符串指针**  **int flag = 1;**  **vector<string> vec;//用来存储输入的算术表达式**  **char a[100] ;//用来存储输入的算术表达式和vec的效果相同，只不过vec中的数字是存储在单个下标中。**  **char fz[100] ; // 符号栈**  **int sz[100] ; // 数字栈**  **int fhead = 0 ; // 符号栈指针**  **int shead = 0 ; // 数字栈指针** |

1. 主要函数

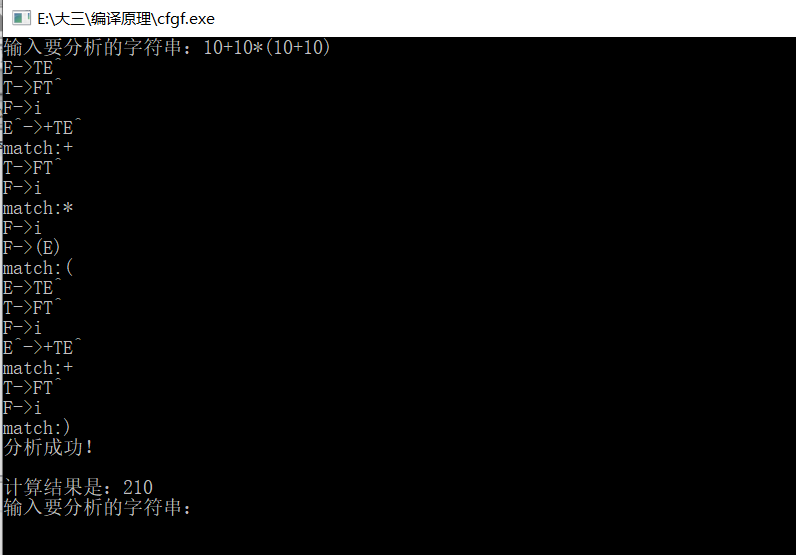
|  |  |
| --- | --- |
| 1 | **void T();**  **void E();**  **void E1();**  **void T1();**  **void F();**  **当输入的非终结符为T，E，E1，T1，F时程序进行的动作。当然程序开始执行时就要执行E()函数，因为他是第一个文法产生式。** |
|  | **举个例子：** |
| 2 | **void E1()**  **{**  **if(str[location]=='+')**  **{**  **a[sum++] = '+';**  **cout<<"E^->+TE^"<<endl;**  **match('+');**  **T();**  **E1();**  **}**  **}**  **因为有文法E1-->+TE^|$,所以此时应该进行的处理就是将+TE^逆序入栈，也就是递归中的顺序执行+（非终结符用来和字符串进行匹配，匹配成功则可以进行之后的递归操作,并且要将字符串的指针往后移一位，否则产生空值），T()和E1()。**  **上述其他的函数类似该函数。** |
|  |  |
| 3 | **void match(char a)用来匹配一个终结符，然后指针后移一位。** |
|  |  |
| 4 | **int invt(char a)用于计算器的实现部分，匹配非数值型字符（即‘（’，‘）’，‘+’，‘\*’），匹配成功返回1，否则返回0** |
|  |  |
| 5 | **void jsq()计算器实现部分**  **void jsq()**  **{**  **int len = strlen(a)-1;**  **for(int i = 0 ; i <= len ; ++ i )**  **{**  **if(a[i] == '(' )//如果匹配到一个（，则进行压栈操作，压入符号栈中**  **{**  **fz[++ fhead] = a[i] ;**  **continue ;**  **}**  **if(a[i]==')')//匹配到），则进行寻找最近的一个（的操作，并且将这两个符号之间的值算出来，压入数字栈。**  **{**  **while(fz[fhead] != '(')**  **mathvt(fz[fhead]) ;**  **--fhead ;**  **continue ;**  **}**  **if(a[i] >= '0' && a[i] <= '9')**  **{**  **++shead ;**  **while(a[i] >= '0' && a[i] <= '9')**  **sz[shead] = sz[shead] \*10 + a[i] - '0' ,i++;**  **i--;**  **continue;**  **}//匹配到一个数字串，转换成int类型压入数字栈。**  **else**  **{**  **if(a[i] == '\*')//匹配到\*直接压栈**  **{**  **fz[++fhead] = a[i];**  **continue;**  **}**  **Else//这里是匹配到+，+的优点度高于\*，所以要计算之前的值**  **while(fz[fhead] == '\*'|| fz[fhead] == a[i])**  **{**  **mathvt(fz[fhead]);**  **}**  **fz[++fhead] = a[i];**  **}**  **}**  **while(fhead != 0) {//当符号栈为空的时候就说明计算完毕。**  **mathvt(fz[fhead]) ;**  **}**  **cout<<endl<<"计算结果是："<<sz[shead]<<endl;**  **fhead = 0;**  **shead = 0;**  **sum = 0;**  **memset(a,'\0',sizeof(a));**  **memset(fz,'\0',sizeof(fz));**  **memset(sz,0,sizeof(sz));**  **}** |

1. **测试与运行情况**
2. 测试样例数据

程序中输入数据如下：

10+10\*（10+10）（可以输入任何只含+和\*运算符的算术表达式，注意不要输入中文括号）

1. 程序执行结果



1. 程序执行结果分析

很明显，计算器输出的结果是正确的，对于LL1分析过程，输出也打印的很清晰，对照输出一步步推导，发现和实际情况相吻合。

1. **实验总结**

1.程序存在的问题

A) 对于CFG的处理由于写死了，没有灵活性，如果要换一个CFG进行判断，则需要将代码重新编写，不够方便，不具有可移植性。

B) 对于抽象语法树的打印没有做到位，问题太多，最后导致没有完成这一功能。

C) 计算器的功能比较简单，只设置了加法和乘法，并且数据量不能够太大，否则会出现溢出。

2. 收获

通过这次实验，对于LL1的分析方法进行了一次加深和巩固，学会了用递归来模拟栈的作用，理解了递归下降分析器的工作过程，明白了为什么递归下降分析每次规约的都是最左直接短语，对于用C语言编写简单计算器有了新的感悟。