算符优先文法实验报告

目录：

1. 实验目的

1.1 实验目的和要求

1）根据算符优先分析法，对表达式进行语法分析，使其能够判断一个表达式是否正确。

2）通过对算法优先分析方法的实现，加深对自下而上语法分析方法的理解。

1.2 实验内容

1) 给定文法，可以是算术表达式的文法是G[E]（可以根据自己的实验进行改变）：

E→E+T| T

T→T*F| F

F→(E)| i

2) 用算法优先分析法对文法G[E]产生的句子进行语法分析，判断该表达式是否正确。

1.3 实验步骤

1) 根据给定文法，先求出FirstVT和LastVT集合；

2) 构造算符优先关系表（要求算符优先关系表输出到屏幕或者输出到文件）；

3) 根据算法和优先关系关系表分析给定表达式是否是该文法识别的正确的算术表达式（要求输出归约过程）；

2. 算法分析

2.1 FirstVt 和 LastVt

按照书本方法计算每个非终结符的FirstVt 和 LastVt， 并存放于文件中准备读入。

2.2 文法存放

文法按照以下格式存放，为了方便在模糊匹配中寻找规约字串，在文法中添加例如 "E->F+F"这样的推导关系；原文法中形如"E->E+T|T"的推导拆分成俩行分别存放；并且存放中同意准寻"A->B*C"改为"AB*C"这样的字符串。

2.3 变量定义

变量定义为代码中注释；

```
string gra[N];//存放文法字符串
map<char, set<char> > firstVt;//FirstVt ,形成非终结符和终结符集合的映射关系
map<char, set<char> > lastVt;//LastVt, 同理
set<char> Vn;//非终结符集合
set<char> Vt;//终结符集合
struct node {
 char a;
 char b;
 int rel; // 优先关系定义 1 为a <. b; 2 为a =. b; 3 为a >. b;
};
map<pair<char, char>, node> compareTbl; // 文法的优先关系表
```

2.4 初始化文法

从文件中读入文法字符串，并且按照大小写区分终结符和非终结符，存放入对应的全局变量中。

```
while(n--) {
cin>>gra[i++];
string s = gra[i-1];
for(int j=0;j<s.length();++j) {
if(s[j]>='A' && s[j]<='Z') {Vn.insert(s[j]);}
```

```
6   else{Vt.insert(s[j]);}
7   }
8   }
```

2.5 计算优先关系表

对于任意终结符a, b , 存在以下定义：

1）当存在推导 A -> ...ab 或者　A -> ...aRb , R ∈ Vn, 则 a =. b

2）当存在推导 A -> ...aR... 并且　R -> b... , R为子串, 则 a <. b

3）当存在推导 A -> ...Rb... 并且 R -> ...a ,R为子串 , 则 a <. b

2.5.1 对于第一种，程序中的滑动窗口算法具体为，对于每一条推导，如果窗口匹配到的第一个为终结符a，则继续向前看，如果第二位为终结符b，则生成一条相等的<a,b>优先关系；如果向前看的第二位为非终结符并且第三位为终结符b, 则生成一条相等的<a,b>优先关系;并且指向下一位的读取指针分别前移。

```
1  //5.6.1 a =. b
2   for(int i=0;i<Len;++i) {
3   string g = gra[i].substr(1);
4   if(g.length()<2) {continue;}
5   for(int j=0;j<g.length();){
6   if(isVt(g[j])) {
7   if(j+1 <g.length() && !isVt(g[j+1])
8   && j+2<g.length() && isVt(g[j+2])){
9   //A -> ...aRb...
10   pair<char, char> p;
11   p.first = g[j];p.second = g[j+2];
12   compareTbl[p] = {g[j], g[j+2], 2};
13   j=j+3;continue;
14   }
15   if(j+1 < g.length() && isVt(g[j+1])) {
16   //A -> ...ab...
17   pair<char, char> p;
18   p.first = g[j];p.second = g[j+1];
19   compareTbl[p] = {g[j], g[j+1], 2};
20   j=j+2;continue;
21   }
22   }
23   j++;
24   }
```

```
25  }
```

2.5.2 对于第二种，程序中的滑动窗口算法具体为，对于每一条推导，如果窗口匹配到的第一个为终结符a，则继续向前看，如果第二位为非终结符R, 则将非终结符对应的FirstVt中的每一个终结符b, 生成小于的优先关系<a, b> 并且存放入优先关系表中。对于第三种类似，只是将匹配到的第一位为非终结符，第二位为终结符a，并且将非终结符对应的LastVt中终结符b 和 a，生成大于的优先关系<b,a>, 并且存放入优先关系表中。

```
// 5.6.2 a <. b
 for(int i=0;i<Len;++i) {
 string g = gra[i].substr(1);
 if(g.length()<2) {continue;}
 for(int j=0;j<g.length();) {
 if(isVt(g[j])) {
 // A -> ...aR , b ∈ FirstVt(R)
 if(j+1 < g.length() && !isVt(g[j+1])) {
 set<char> fvt = firstVt[g[j+1]];
  set<char>::iterator it;
  char a = g[j];
  for (it=fvt.begin() ; it != fvt.end();
  it++ ) {
  char b = *it;
  pair<char, char> p;
  p.first = a;p.second = b;
  compareTbl[p] = {a, b, 1};
  }
  j = j+2;continue;
  }
  }
  j++;
  }
  }
```

2.6 打印优先关系表

循环输出所有终结符对应的优先关系，并且约定 '#' <. a, 并且 a >. '#'， a ∈Vt;

```
# < (
# < )
# < *
# < +
```

```
# < i
( > #
( < (
( = )
( < *
 ( < +
 ( < i
 ) > #
 ) > )
 ) > *
 ) > +
 * > #
 * < (
 * > )
 * > *
 * > +
 * < i
 + > #
 + < (
 + > )
 + > +
 + < i
 i > #
 i > )
 i > *
 i > +
```

2.7 对于表达式进行规约

初始化一个栈，并且存放入 '#'，poschar 存放栈顶或者次栈顶的终结符， 每次循环读入当前表达式的下一个字符a，如果a的优先级大于或者等于poschar 的优先级，则直接将a 压入栈中，否则不读入当前字符a, 并且对于栈中元素进行规约。

规约方法：由于进行规约的前提是下一个读入字符的优先级比poschar 小，表示找到了可规约字符串的尾部，为了找到改字符的规约头部，即找到第一个出现栈中元素c, 而c 的优先级 <. poschar 的优先级，过程为循环弹出栈顶元素，并进行比较，并生成可规约字符串，并在文法中模糊匹配对应的推导非终结符，如果能找到，则将对应的非终结符重新压栈，否则不存在对应推导文法，并报错，退出程序。

```
void check2(string s) {
 stack<char> stk;
```

```
stk.push('0');
stk.push('#');
bool flag = true;
int i=0; //s[i]
while(flag) {
char a = s[i++];
char poschar;
 if(stk.top()=='#' || isVt(stk.top())) {poschar = stk.top();}
 else{
 char temp = stk.top();stk.pop();
 poschar = stk.top();stk.push(temp);
 }
 if(a =='#') {flag = false;}
 pair<char, char> p;p.first = poschar;p.second = a;
 int prim = compareTbl[p].rel;
 if(prim == 1 || prim == 2) {stk.push(a);}
 else if(prim == 3) {
 i--;
 //规约
 string s1 = "";
 if(poschar != stk.top()) {
 s1 += stk.top();stk.pop();
 s1 += stk.top();stk.pop();
 }
 if(poschar == stk.top()) {s1 += poschar;stk.pop();}
 while(1) {
 char pp = stk.top();
 pair<char, char> p;
 p.first = pp;p.second = poschar;
 int prim2 = compareTbl[p].rel;
 if(prim2 == 1 ) {break;}
 else{
 s1 += pp;stk.pop();
 if(isVt(pp)) {poschar = pp;}
 }
 }
 string s2 = "";
 s2.assign(s1.rbegin(), s1.rend());
 char newHead = findMergeStr(s2);
 if(newHead == '~' ) {cout<<"wrong..."<<endl;return ;}
```

```
43  stk.push(newHead);
44  cout<<newHead<<" -> "<<s2<<endl;
45  }
46  else {cout<<"error..."<<endl;return ;}
47  string con = getStkStr(stk);
48  cout<<"now stk string: "<<con<<endl;
49  }
50  //最后栈中 # ... #
51  string tmps = "";
52  while(stk.top()!='#') {tmps += stk.top();stk.pop();}
53  string rs = "";
54  rs.assign(tmps.rbegin(), tmps.rend());
55  char newHead = findMergeStr(rs);
56  stk.push(newHead);
57  cout<<newHead<<" -> "<<rs<<endl;
58  string con = getStkStr(stk);
59  cout<<"now stk string: "<<con<<endl;
60  if(flag==false) {cout<<"analysis complete,process is right..."<<endl;}
```

2.8 程序结果验证

2.8.1 表达式 "(i+i)*i" √

```
1  analysis string :(i+i)*i#
2  now stk string: 0#(
3  now stk string: 0#(i
4  F -> i
5  now stk string: 0#(F
6  now stk string: 0#(F+
7  now stk string: 0#(F+i
8  F -> i
9  now stk string: 0#(F+F
10  E -> F+F
11  now stk string: 0#(E
12  now stk string: 0#(E)
13  F -> (E)
14  now stk string: 0#F
15  now stk string: 0#F*
16  now stk string: 0#F*i
17  F -> i
```

```
now stk string: 0#F*F
E -> F*F
now stk string: 0#E
analysis complete, process is right...
```

2.8.2 表达式 "((i+i)+(i+i))*i" √

```
analysis string :((i+i)+(i+i))*i#
now stk string: 0#(
now stk string: 0#((
now stk string: 0#((i
F -> i
now stk string: 0#((F
now stk string: 0#((F+
now stk string: 0#((F+i
F -> i
now stk string: 0#((F+F
E -> F+F
now stk string: 0#((E
now stk string: 0#((E)
F -> (E)
now stk string: 0#(F
now stk string: 0#(F+
now stk string: 0#(F+(
now stk string: 0#(F+(i
F -> i
now stk string: 0#(F+(F
now stk string: 0#(F+(F+
now stk string: 0#(F+(F+i
F -> i
now stk string: 0#(F+(F+F
E -> F+F
now stk string: 0#(F+(E
now stk string: 0#(F+(E)
F -> (E)
now stk string: 0#(F+F
E -> F+F
now stk string: 0#(E
now stk string: 0#(E)
F -> (E)
now stk string: 0#F
```

```
now stk string: 0#F*
now stk string: 0#F*i
F -> i
now stk string: 0#F*F
E -> F*F
now stk string: 0#E
analysis complete, process is right...
```

2.8.3 表达式 "i+i)*i" ×

```
analysis string :i+i)*i#
now stk string: 0#i
F -> i
now stk string: 0#F
now stk string: 0#F+
now stk string: 0#F+i
F -> i
now stk string: 0#F+F
E -> F+F
now stk string: 0#E
now stk string: 0#E)
find no merge str, process is wrong...
```

3. 实验感想和改进

1）程序中为了方便，直接将FirstVt和LastVt存入文件中，并没有按照文法规则进行推导，可以尝试不断改进和完善程序。

2）规约中的模糊匹配应该按照推导中非终结符位置进行匹配，而文中提前将可能的推导写入文法中，虽然不影响，但是值得程序实现和改进模糊匹配算法。

3）此次程序明显与之前的三次程序实现难度增大，不过受益颇多，进一步掌握了优先关系表的推导和生成过程，并且对规约的理解更加深入。