目录

[一、 前言 2](#_Toc106800026)

[二、 开发过程 2](#_Toc106800027)

[（一） 设计思路 2](#_Toc106800028)

[（二） 实现过程 2](#_Toc106800029)

[三、 设计方案 12](#_Toc106800030)

[（一） 功能组成 12](#_Toc106800031)

[（二） 界面组成 12](#_Toc106800032)

[（三） 例子结果展示 14](#_Toc106800033)

[（四） 代码实现 17](#_Toc106800034)

[1.保存文法规则 17](#_Toc106800035)

[2.打开文法规则 18](#_Toc106800036)

[3. 求两个vector的并集 18](#_Toc106800037)

[4.消除有害规则和多余规则 18](#_Toc106800038)

[5.消除左递归（包括直接和间接） 23](#_Toc106800039)

[6.提取左公因子（包括直接和间接） 29](#_Toc106800040)

[7. 求first集合 39](#_Toc106800041)

[8.求follow集合 40](#_Toc106800042)

[9.实现最左推导分析 42](#_Toc106800043)

[10.退出系统 45](#_Toc106800044)

# 前言

本次实验是实现一个文法问题处理器，可以通过用户输入正确的文法规则，然后消除文法规则中的有害规则和多余规则，然后再删除左递归，接着再删除左公因子，最后将化简后的文法规则求出first集合和follow集合，然后就可以让用户输入符号串（以$号结尾）判断是否输入正确，如果符号串错误将提示错误。

# 开发过程

## 设计思路

1. 首先完成页面布局
2. 保存，打开存有文法规则的功能按钮的实现
3. 实现删除文法规则中的有害规则和多余规则
4. 实现消除文法规则中的左递归（包括直接和间接）
5. 实现文法规则中的提取左公因子（包括直接和间接）
6. 对上述消除左递归和提取完左公因子后的文法再进行一次删除多余规则
7. 实现将上述步骤的化简后求first集合和follow集合
8. 实现对输入的句子进行最左推导分析
9. 退出系统

## 实现过程

1. 界面设计如下：











1. 保存，打开存有文法规则的功能按钮的实现

使用Qt的文件QfileDialog、QFile和QTextStream实现保存和打开功能。

fileDialog.getSaveFileName()函数实现获取保存文件的路径，再使用QFile和QTextStream将文法规则写入文件。

QFileDialog::getOpenFileName()函数实现获取打开文件的路径名，再使用

QFile和QTextStream将文法规则读取出来并显示在编辑框上。

1. 实现删除文法规则中的有害规则和多余规则

当文法左右两边为U->U的形式进行删除有害规则

多余规则：删除不可终止的文法规则，首先以终结符号集合为起点，然后再创建一个空集合T，如果文法规则中存在从非终结符号推出的全部右部符号在终结符号集合或者S中，则将该非终结符号加入S，直到S不在变化即删除了不可终止的文法。删除不可到达的文法规则，在前面删除不可终止的基础上，创建包含文法开始符号集合T，从文法开始符号开始，找出有集合T推出的文法规则加入到集合T中，在集合T中找非终结符号将其推出的右部加入集合T中直到集合不发生改变即结束，删除不可到达的文法规则。

**例如**：S -> AC | B

A -> a

C -> c | BC

E -> aA | e

非终结符号集合N为｛a,c,e｝

空集合T：｛｝

第一轮：S->AC该文法右部不在N或T中

S->B 该文法右部不在N或T中

A->a 该文法的右部在N中，加入T,此时T为｛A｝

C->c 该文法的右部在N中，加入T,此时T为｛A,C｝

C->BC 该文法的右部符号B不在N或T中

E->aA 该文法的右部在N和T中，加入T,此时T为｛A,C,E｝

E->e 该文法的右部在N中，加入T,此时T为｛A,C,E｝

第二轮：S->AC该文法右部在T中，加入T,此时T为｛A,C,E,S｝

S->B 该文法右部不在N或T中

A->a 该文法的右部在N中，加入T,此时T为｛A,C,E,S｝

C->c 该文法的右部在N中，加入T,此时T为｛A,C,E,S｝

C->BC 该文法的右部符号B不在N或T中

E->aA 该文法的右部在N和T中，加入T,此时T为｛A,C,E,S｝

E->e 该文法的右部在N中，加入T,此时T为｛A,C,E,S｝

第三轮：……

第三轮后T集合没有变化即终止循环

将规则中存在B的文法规则删除掉

删除不可到达：T={S}

S -> AC

A -> a

C -> c

E -> aA | e

第一轮：S->AC,将AC加入T，T{S,A,C}

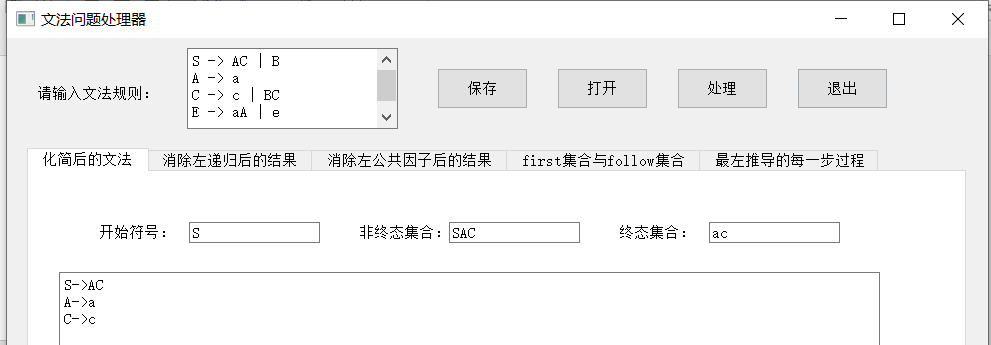
第二轮：S->AC, 将AC加入T，T{S,A,C}

A->a,将a加入T，T{S,A,C,a}

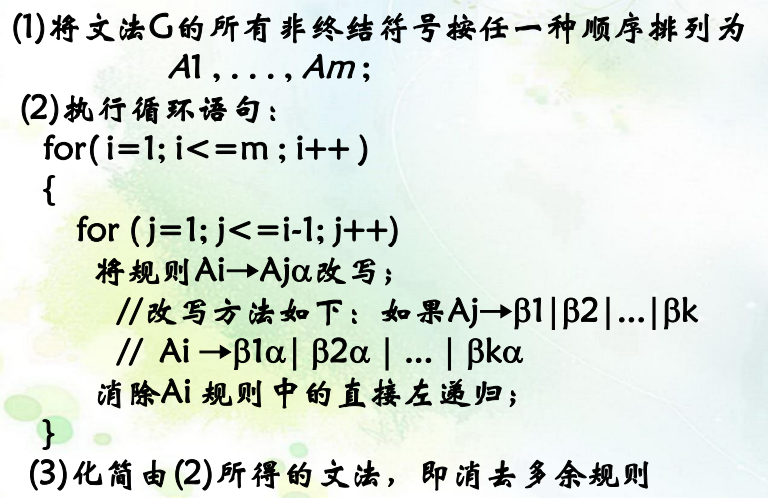
C->c,将c加入T，T{S,A,C,c}

第三轮后T集合没有变化即终止循环

将规则中不在T集合的文法规则删除掉

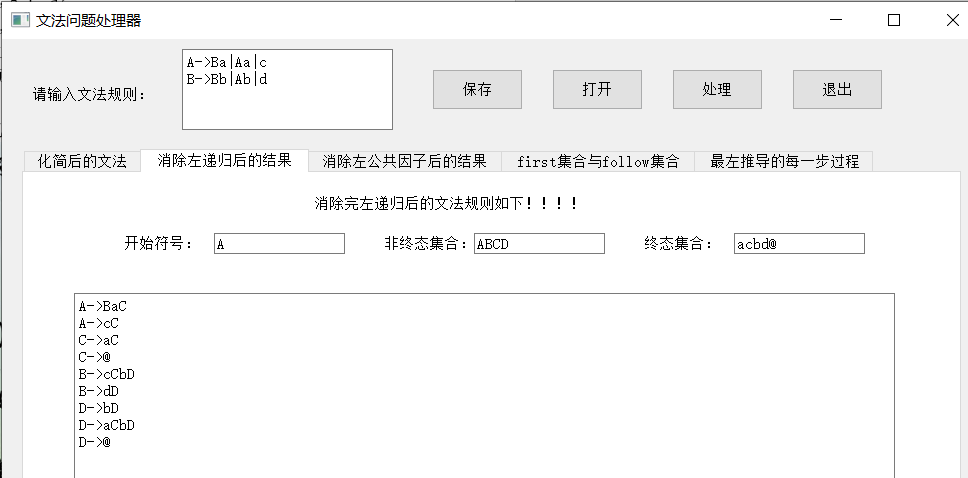


1. 实现消除文法规则中的左递归（包括直接和间接）



每次改造文法产生的新非终结符号都是用未用过的大写字母

例子：



1. 实现文法规则中的提取左公因子（包括直接和间接）

首先先求出各个非终结符号和终结符号的first集合,其次，求每个非终结符号的左公因子的提取，找到一个非终结符号为左部开始的文法规则，将其所有右部的文法规则的第一个符号求first，并将其first集合求交集为T，求交集后的这个集合就是所有可能出现左公因子的首个符号，然后循环该T集合，每一次循环取出一个元素，并将其右部规则中首个符号的first集合包含该元素的放在一起，代表这些规则有相同的左部，直到循环结束就会将其按以集合T元素开头的各个集合，用Map来保存,一个元素对应着首字符的first集合有该元素的所有规则的集合，然后遍历循环该Map，当集合大于1的说明有公因子则去提取公因子，集合小于1的说明以该符号没有左公因子。提取左公因子时，如果是间接左公因子将会找到该符号的右部规则并将其带回。

**例如：**S->cdAd | cdB

A->ab | a

B->aba

求S的左公因子：首先求两条规则首字符first集合的交集，即T{c},接着找到以c开头或者间接开头的规则，发现有两条规则S->cdAd | cdB，然后扫描c后面的符号，如果有相同的则加入左公因子，即左公因子为cd，改造文法，

S->cdC,C->Ad|B

求A的左公因子：左公因子为a，改造文法后为：A->aD,D->b|@;

求B的左公因子：左公因子a只有一条规则，说明该符号没有左公因子，不用改造,B->aba

求C的左公因子：C->Ad|B,首先求得两条规则首字符first集合的交集为T{a},然后去找左公因子a，由于是间接的左公因子，所以将A和B对应的文法规则带回，即C->aD|aba,提取左公因子，改造文法后为：C->aE,E->Dd|ba;

求D的左公因子：没有，即不用改造文法规则，D->b|@

求E的左公因子：E->Dd|ba，首先求得两条规则首字符first集合的交集为T{b,@},然后以b为左公因子的文法规则大于1，所以提取左公因子，由于是间接的左公因子，所以将D对应的右部规则带回，即E->bd|d|ba,提取左公因子b，改造文法规则为：E->bF,F->d|a

求F的左公因子：没有左公因子，不用改造, F->d|a

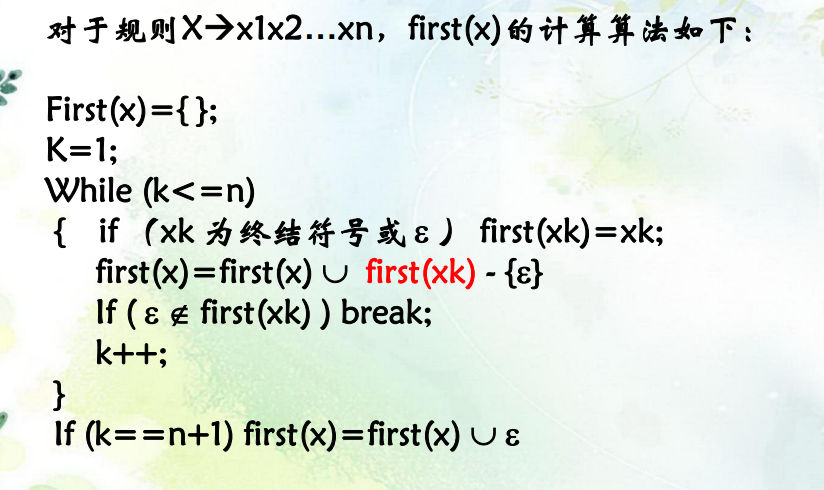
综上,求完左公因子后的文法规则为：



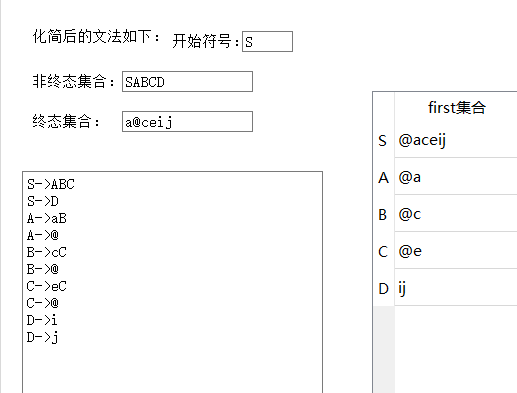
1. 对上述消除左递归和提取完左公因子后的文法再进行一次删除多余规则

过程与上面的第3点一样

1. 实现将上述步骤的化简后求first集合和follow集合



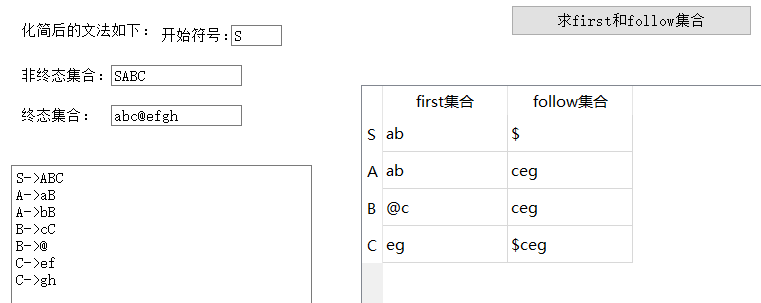
first集合的算法是课件的算法，求first()函数用递归实现



follow集合的算法也是课本的算法，如下图：



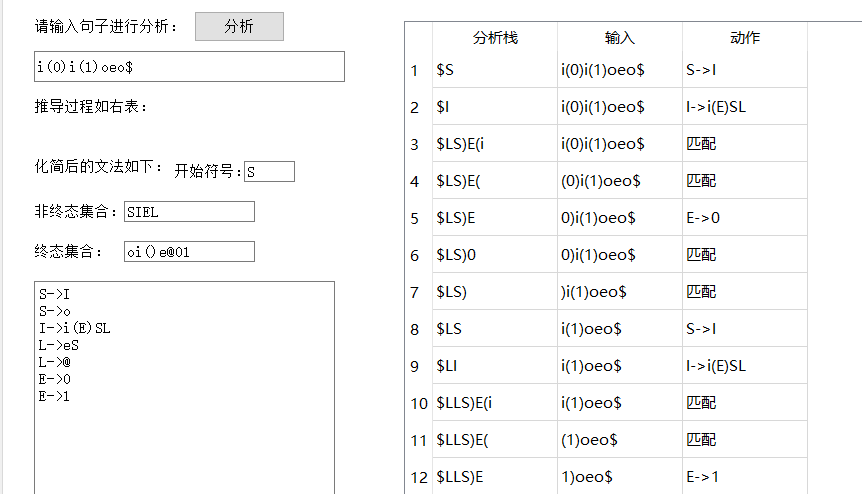
例子：



1. 实现对输入的句子进行最左推导分析

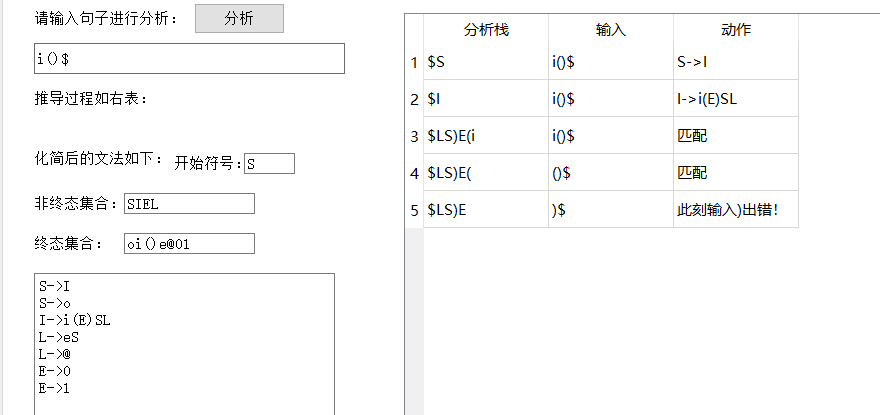
最左推导采用LL(1),首先将$和开始符号压入分析栈，然后，从分析栈弹出一个字符ch，若ch与输入的句子的首个字符相同则匹配，删除输入句子的首个字符，判断下一个字符，如果ch是非终结符号且求其first集合包含输入的句子的首个字符，如果ch的follow集合也包含该首个字母并且ch存在文法规则ch->@并且输入的句子的下一个字母是$时，则使用文法规则ch->@，不然就是找ch开头的文法规则的右部的首字符包含输入句子的首个字符的文法规则的右部反向压入分析栈；如果ch是非终结符且其first集合不包含输入的句子的首个字符，如果ch的follow包含输入的句子的首字母且存在文法规则ch->@，则使用文法规则ch->@，如果不存在文法规则ch->@，则输出错误；如果ch是非终结符且其first集合和follow集合都不包含输入的句子的首个字符，则输出错误，句子错误，依次类推直到$和$匹配则为接受状态。

**接受状态的例子：**





**出错的例子：**



1. 退出按钮功能

exit(0)

# 设计方案

## 功能组成

1. 保存文法规则
2. 打开文法规则
3. 求两个vector的并集
4. 消除有害规则和多余规则
5. 消除左递归（包括直接和间接）
6. 提取左公因子（包括直接和间接）
7. 求first集合
8. 求follow集合
9. 实现最左推导分析
10. 退出系统

## 界面组成







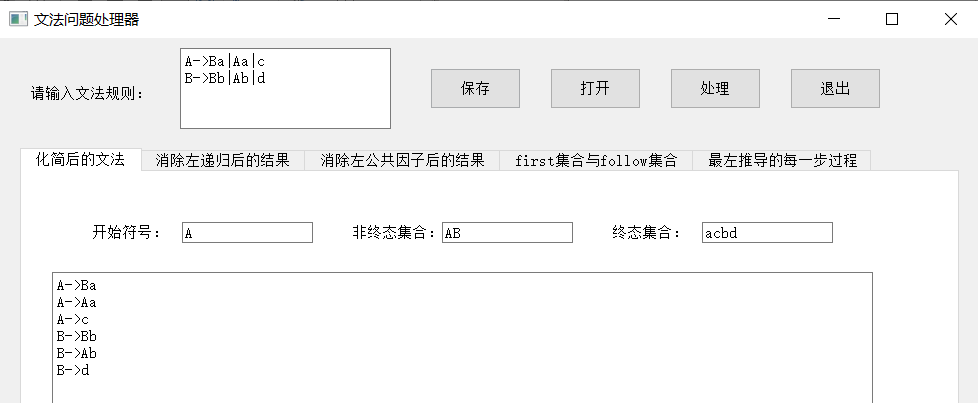


## 例子结果展示

eg. A->Ba|Aa|c

B->Bb|Ab|d

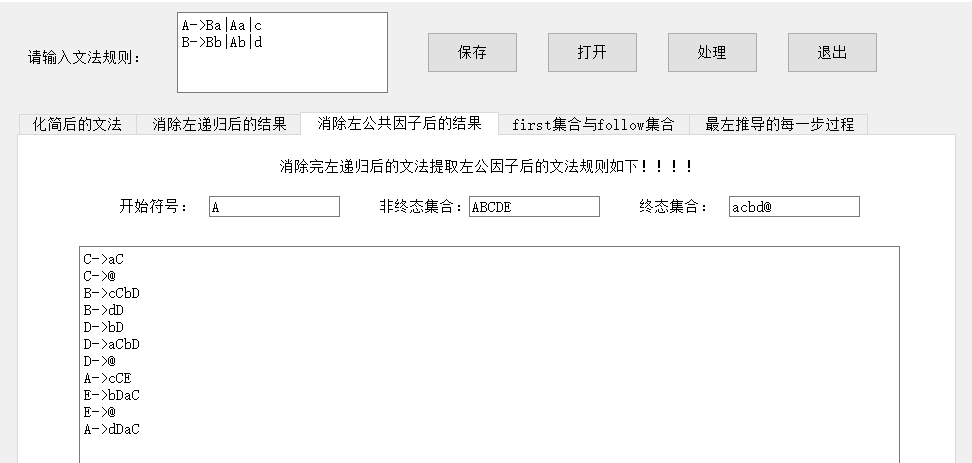
删除多余规则和有害规则：



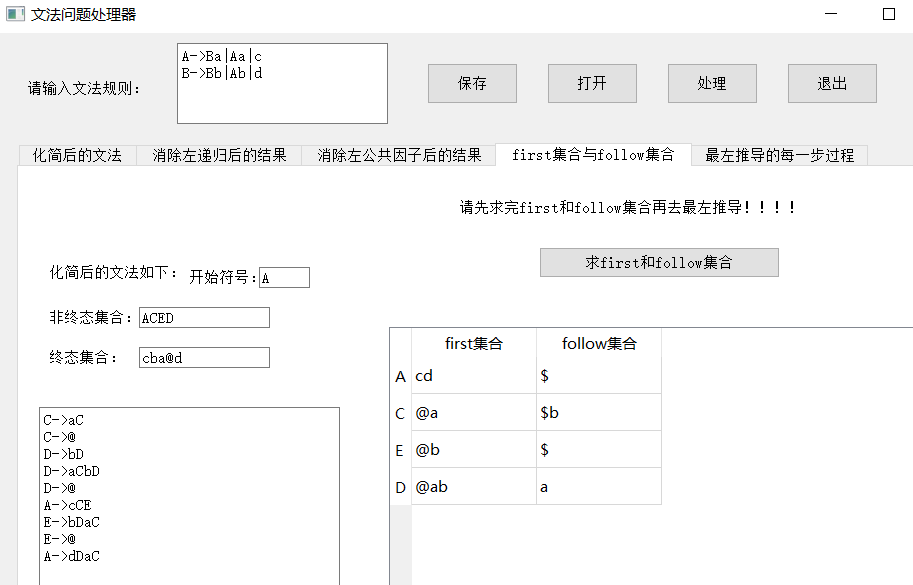
消除左递归（以删除多余规则和有害规则后的文法去消除左递归）：



提取左公因子（以消除完左递归后的文法去提取左公因子）：



求first和follow集合



最左推导分析：



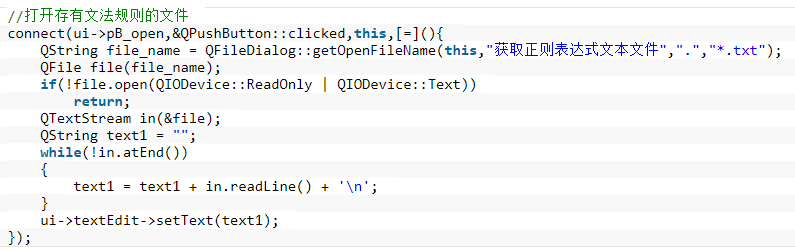


## 代码实现

### 1.保存文法规则



### 2.打开文法规则



### 3. 求两个vector的并集

//两个vector求并集

QVector<QChar> MainWindow::**vectors\_set\_union**(QVector<QChar> vt1,QVector<QChar> vt2){

QVector<QChar> v;

sort(vt1.begin(),vt1.end());

sort(vt2.begin(),vt2.end());

set\_union(vt1.begin(),vt1.end(),vt2.begin(),vt2.end(),back\_inserter(v)); //求交集

return v;

}

### 4.消除有害规则和多余规则

void MainWindow::**simplify**() //删除多余规则

{

QVector<QChar> temp1,temp2,temp3; //存放可以推到出到终结符号的非终结符号

temp1 = t1.terminalSet;

while(true) //消除不可终止

{

textNode \*p = t1.textG;

temp3 = temp2;

while(p!= NULL) //判断文法规则是否结束

{

rightNode \*p1 = p->right;

while(p1!=NULL)

{

if(temp1.indexOf(p1->rightCh) == -1 && temp2.indexOf(p1->rightCh) == -1)

break;

p1 = p1->next;

}

if(p1 == NULL)

{

if(temp2.indexOf(p->leftCh) == -1)

temp2.append(p->leftCh);

}

p = p->next;

}

if(temp2 == temp3)

break;

}

textNode \*p = t1.textG;

textNode \*q = t1.textG; //文法的链表结构的链接

while(p!= NULL) //删掉不可终止的文法规则

{

if(temp2.indexOf(p->leftCh) == -1)

{

if(q==p)

{

q = p->next;

while(p->right!=NULL)

{

rightNode \*p1 = p->right;

p->right = p->right->next;

delete p1;

}

delete p;

p = q;

t1.textG = p;

continue;

}

q->next = p->next;

while(p->right!=NULL)

{

rightNode \*p1 = p->right;

p->right = p->right->next;

delete p1;

}

delete p;

p = q->next;

}

else

{

rightNode \*p1 = p->right;

while(p1!=NULL)

{

if(p1->rightCh >= 65 && p1->rightCh <= 90 && temp2.indexOf(p1->rightCh) == -1)

{

if(q==p)

{

q = p->next;

while(p->right!=NULL)

{

rightNode \*p1 = p->right;

p->right = p->right->next;

delete p1;

}

delete p;

p = q;

t1.textG = p;

break;

}

q->next = p->next;

while(p->right!=NULL)

{

rightNode \*p1 = p->right;

p->right = p->right->next;

delete p1;

}

delete p;

p = q->next;

break;

}

p1 = p1->next;

}

if(p1 == NULL)

{

q = p;

p = p->next;

}

}

}

t1.non\_terminalSet.clear();

t1.non\_terminalSet = temp2; //将不可终止的消除后的非终结符号的集合

temp1.clear();

temp2.clear();

temp1.append(t1.start);

while(true) //消除不可到达

{

textNode \*p = t1.textG;

temp2 = temp1;

while(p!=NULL)

{

if(temp1.indexOf(p->leftCh) != -1)

{

rightNode \*p1 = p->right;

while(p1!=NULL)

{

if(temp1.indexOf(p1->rightCh) == -1)

{

temp1.append(p1->rightCh);

}

p1 = p1->next;

}

}

p = p->next;

}

if(temp1 == temp2)

break;

}

//先将文法规则的终结符号集合和非终极符号集合清空，以边存入化简后的新集合

t1.non\_terminalSet.clear();

t1.terminalSet.clear();

for(int i=0;i<temp1.size();i++)

{

if(temp1[i]>=65 && temp1[i]<=90)

t1.non\_terminalSet.append(temp1[i]);

else

t1.terminalSet.append(temp1[i]);

}

p = t1.textG;

q = t1.textG; //文法的链表结构的链接

while(p!= NULL) //删掉不可到达的文法规则

{

if(temp2.indexOf(p->leftCh) == -1)

{

if(q==p)

{

q = p->next;

while(p->right!=NULL)

{

rightNode \*p1 = p->right;

p->right = p->right->next;

delete p1;

}

delete p;

p = q;

t1.textG = p;

continue;

}

q->next = p->next;

while(p->right!=NULL)

{

rightNode \*p1 = p->right;

p->right = p->right->next;

delete p1;

}

delete p;

p = q->next;

}

else

{

rightNode \*p1 = p->right;

while(p1!=NULL)

{

if(p1->rightCh >= 65 && p1->rightCh <= 90 && temp2.indexOf(p1->rightCh) == -1)

{

if(q==p)

{

q = p->next;

while(p->right!=NULL)

{

rightNode \*p1 = p->right;

p->right = p->right->next;

delete p1;

}

delete p;

p = q;

t1.textG = p;

break;

}

q->next = p->next;

while(p->right!=NULL)

{

rightNode \*p1 = p->right;

p->right = p->right->next;

delete p1;

}

delete p;

p = q->next;

break;

}

p1 = p1->next;

}

if(p1 == NULL)

{

q = p;

p = q->next;

}

}

}

if(t1.textG==NULL)

{

t1.non\_terminalSet.clear();

t1.terminalSet.clear();

t1.start = '\0';

}

}

### 5.消除左递归（包括直接和间接）

其中需要调用的函数为：

QVector<QVector<QChar>> MainWindow::**findRight**(QChar ch,QChar left) //找非终结符号ch的所有右部并返回，left用于查看右部是否包含这个字母，以便决定是否要返回右部规则

{

QVector<QVector<QChar>> right;

textNode \*p = t1.textG;

bool flag = false; //判断是否要返回

while (p!=NULL) {

if(p->leftCh == ch){

QVector<QChar> temp;

rightNode \*p1 = p->right;

if(p1->rightCh == left)

flag = true;

while(p1!=NULL)

{

temp.append(p1->rightCh);

p1=p1->next;

}

right.append(temp);

}

p= p->next;

}

if(flag)

return right;

else

{

right.clear();

return right;

}

}

void MainWindow::**deleteLeftRecursion**() //消除左递归

{

//消除左递归，非终结符号按存放非终态集合的顺序排列t1.non\_terminalSet

int length = t1.non\_terminalSet.size();

for(int i=0;i<length;i++)

{

QVector<QVector<QChar>> rightRule; //存放所有右部规则，便于查看哪些右部规则是左递归

QVector<QVector<QChar>> sameLeft; //存放是左递归的文法规则右部

QVector<QChar> cleanCh; //存放已被消除左递归的非终结符号

QMap<QChar,int> flag; //用于代表前面已经消除左递归的非终结符号是否包含间接左递归，若包含则为，否则为0

for(int j=0;j<=i-1;j++)

{

cleanCh.append(t1.non\_terminalSet[j]);

flag[t1.non\_terminalSet[j]] = 0;

}

int n =0; //用于记录原先左部为t1.non\_terminalSet[i]的前n条规则，便于后面删除

textNode \*p=t1.textG;

textNode \*q;

while(p!=NULL)

{

if(p->leftCh == t1.non\_terminalSet[i])

{

n++;

QVector<QChar> temp;

rightNode \*p1 = p->right;

if(cleanCh.indexOf(p1->rightCh)!=-1)

{

QVector<QVector<QChar>> tempRight = this->findRight(p1->rightCh, t1.non\_terminalSet[i]);

if(tempRight.empty()) //若返回为空，说明改规则不存在间接左递归，无须将右部带回规则中

{

temp.clear();

while(p1!=NULL)

{

temp.append(p1->rightCh);

p1=p1->next;

}

rightRule.append(temp);

p = p->next;

continue;

}

flag[p1->rightCh] = 1;

for(int j =0;j<tempRight.size();j++)

{

temp.clear();

rightNode \*q1 = p1->next;

for(int k=0;k<tempRight[j].size();k++)

{

temp.append(tempRight[j][k]);

}

while(q1!=NULL)

{

temp.append(q1->rightCh);

q1=q1->next;

}

rightRule.append(temp);

}

}

else

{

temp.clear();

while(p1!=NULL)

{

temp.append(p1->rightCh);

p1=p1->next;

}

rightRule.append(temp);

}

}

if(p->next==NULL)

q=p;

p = p->next;

}

for(int j=0;j<rightRule.size();j++) //把右部分为包含左递归和非左递归

{

if(rightRule[j][0]==t1.non\_terminalSet[i])

{

rightRule[j].removeOne(t1.non\_terminalSet[i]);

sameLeft.append(rightRule[j]);

rightRule.remove(j);

j--;

}

}

if(sameLeft.empty())

continue;

for(int j=0;j<rightRule.size();j++) //将右部不包含左递归的构建以t1.non\_terminalSet[i]为左部的文法规则加入文法结构中

{

if(!rightRule[j].empty())

{

textNode \*s = new textNode(t1.non\_terminalSet[i]);

s->right = new rightNode(rightRule[j][0]);

rightNode \*p1 = s->right;

for(int k=1;k<rightRule[j].size();k++)

{

rightNode \*q1 = new rightNode(rightRule[j][k]);

p1->next = q1;

p1 = q1;

}

rightNode \*q1 = new rightNode(upperCase[0]);

p1->next = q1;

p1=q1;

p1->next=NULL;

q->next=s;

q=s;

}

}

q->next=NULL;

for(int j=0;j<sameLeft.size();j++) //将右部包含左递归的构建以upperCase[0]为左部的文法规则加入文法结构中

{

if(!sameLeft[j].empty())

{

textNode \*s = new textNode(upperCase[0]);

s->right = new rightNode(sameLeft[j][0]);

rightNode \*p1 = s->right;

for(int k=1;k<sameLeft[j].size();k++)

{

rightNode \*q1 = new rightNode(sameLeft[j][k]);

p1->next = q1;

p1 = q1;

}

rightNode \*q1 = new rightNode(upperCase[0]);

p1->next = q1;

p1=q1;

p1->next=NULL;

q->next=s;

q=s;

}

}

textNode \*s = new textNode(upperCase[0]); //加上文法规则upperCase[0]->@

if(t1.terminalSet.indexOf('@')==-1)

t1.terminalSet.append('@');

s->right = new rightNode('@');

s->right->next =NULL;

q->next = s;

q = s;

q->next=NULL;

judgeLeft[upperCase[0]] = 0;

t1.non\_terminalSet.append(upperCase[0]);

upperCase.removeFirst(); //删除已用的非终结符

p = t1.textG; //将以t1.non\_terminalSet[i]为左部的文法规则前n条规则删除掉

int n1 = 0;

q = p;

while (p!=NULL) { //将以t1.non\_terminalSet[i]为左部的文法规则前n条规则删除掉

if(p->leftCh==t1.non\_terminalSet[i])

{

if(q == p)

{

q = p->next;

delete p;

p = q;

t1.textG = p;

n1++;

if(n1==n)

break;

continue;

}

else

{

q->next = p->next;

delete p;

p = q->next;

n1++;

if(n1==n)

break;

continue;

}

n1++;

if(n1==n)

break;

}

q = p;

p = p->next;

}

}

//输出存储结构中的消除左递归后的文法规则

textNode \*p =t1.textG;

QString str = "";

while(p!=NULL)

{

rightNode \*p1 = p->right;

str = str + p->leftCh + "->";

while(p1!= NULL)

{

str = str + p1->rightCh;

p1= p1->next;

}

str = str + "\n";

p = p->next;

}

ui->textEdit\_4->setText(str);

str = "";

str = str + t1.start;

ui->lineEdit\_start3->setText(str);

str = "";

for(int i=0;i<t1.non\_terminalSet.size();i++)

{

str = str +t1.non\_terminalSet[i];

}

ui->lineEdit\_non3->setText(str);

str = "";

for(int i=0;i<t1.terminalSet.size();i++)

{

str = str +t1.terminalSet[i];

}

ui->lineEdit\_te3->setText(str);

}

### 6.提取左公因子（包括直接和间接）

QVector<QChar> MainWindow::**findTextRule**(QChar leftCh,QChar rightFirst, bool f,QChar lf) //求左部规则为leftCh，右部规则的第一个字符为rightFirst的右部文法规则

{

if(judgeLeft[leftCh] != 1) //若该非终结符号还没提取左公因子就先提取

this->leftFactorChar(leftCh);

textNode \*p = t1.textG;

QVector<QChar> temp;

while(p!=NULL)

{

if(p->leftCh==leftCh)

{

rightNode \*p1 = p->right;

if(p1->rightCh != rightFirst && p1->rightCh !=lf)

{

p = p->next;

continue;

}

while(p1!=NULL)

{

temp.append(p1->rightCh);

p1 = p1->next;

}

break;

}

p = p->next;

}

if(p!=NULL)

{

return temp;

}

else

{

p = t1.textG;

temp.clear();

while(p!=NULL)

{

if(p->leftCh==leftCh)

{

rightNode \*p1 = p->right;

if(t1.non\_terminalSet.indexOf(p1->rightCh) != -1)

{

if(firstMap[p1->rightCh].indexOf(rightFirst) != -1)

{

if(f)

{

temp = findTextRule(p1->rightCh,rightFirst,true,lf);

p1 = p1->next;

while(p1!=NULL)

{

temp.append(p1->rightCh);

p1 = p1->next;

}

}

else

{

while(p1!=NULL)

{

temp.append(p1->rightCh);

p1 = p1->next;

}

}

break;

}

}

}

p = p->next;

}

return temp;

}

}

void MainWindow::**leftFactorChar**(QChar ch) //提取某个具体的非终结符号的左公因子

{

QMap<QChar,QVector<int>> tempStatus; //记录哪几条文法规则有相同的左公因子，以在文法规则中出现的序号(以序号0为开始计数）为记录依据

QVector<QVector<QChar>> saveRightFirst; //存放右部规则的第一个字母的first集合

QVector<QChar> tempLeft; //记录所有文法规则可能出现的最左公因子

textNode \*p = t1.textG;

int i=0;

while(p!=NULL)

{

if(p->leftCh == ch)

{

if(firstMap[p->right->rightCh].empty())

qDebug()<<"为空！";

saveRightFirst.append(firstMap[p->right->rightCh]);

i++;

}

p = p->next;

}

bool \*save = new bool[i];

for(int j =0;j<i;j++)

save[j] = true; //true代表第几条左部规则为ch的文法规则需要删掉

for(int j=0;j<saveRightFirst.size();j++)

tempLeft = this->vectors\_set\_union(tempLeft,saveRightFirst[j]);

for(int j=0;j<tempLeft.size();j++)

{

for(int k=0;k<saveRightFirst.size();k++)

{

if(saveRightFirst[k].indexOf(tempLeft[j]) != -1)

tempStatus[tempLeft[j]].append(k);

}

}

for(int j = 0;j<tempLeft.size();j++)

{

if(tempStatus[tempLeft[j]].size()>1)

{

QVector<QVector<QChar>> findLeftT; //存放具有相同左公因子的文法规则的右部

QVector<QChar> left; //存放左公因子

left.append(tempLeft[j]);

int n = 0; //记录文法规则左部为ch在文法中的位置

p = t1.textG;

textNode \*q;

while (p!=NULL) {

if(p->leftCh==ch)

{

if(tempStatus[tempLeft[j]].indexOf(n) !=-1)

{

QVector<QChar> temp;

rightNode \*p1 = p->right;

if(t1.non\_terminalSet.indexOf(p1->rightCh) != -1 && p1->rightCh!=ch)

{

temp = findTextRule(p1->rightCh,tempLeft[j],true,ch);

}

else

{

temp.append(p1->rightCh);

}

p1 = p1->next;

while(p1!=NULL)

{

temp.append(p1->rightCh);

p1 = p1->next;

}

findLeftT.append(temp);

}

n++;

}

if(p->next==NULL)

q = p;

p = p->next;

}

for(int k =1;k<findLeftT[0].size();k++) //用于求右部规则剩下的符号中有没有存在相同的左公因子

{

bool flag = true;

for(int k1=1;k1<findLeftT.size();k1++)

{

if(findLeftT[k1].size()>k)

{

if(findLeftT[0][k] != findLeftT[k1][k])

{

flag = false;

break;

}

}

else

{

flag = false;

break;

}

}

if(flag)

left.append(findLeftT[0][k]); //存在则加入左公因子集合

else

break;

}

for(int k1 = 0;k1<left.size();k1++) //删除右部规则的相同的左公因子，便于改造文法规则

{

for(int k=0;k<findLeftT.size();k++)

{

findLeftT[k].removeOne(left[k1]);

}

}

//将改造后的文法加入的文法规则的末尾

textNode \*s = new textNode(ch);

s->right = new rightNode(left[0]);

rightNode \*s1 = s->right;

for(int k =1; k<left.size();k++)

{

rightNode \*q1 = new rightNode(left[k]);

s1->next = q1;

s1 = q1;

}

rightNode \*q1 = new rightNode(upperCase[0]);

s1->next = q1;

s1 = q1;

s1->next=NULL;

q->next = s;

q = s;

t1.non\_terminalSet.append(upperCase[0]);

judgeLeft[upperCase[0]] = 0;

for(int k=0;k<findLeftT.size();k++)

{

s = new textNode(upperCase[0]);

if(findLeftT[k].empty())

{

if(t1.terminalSet.indexOf('@')==-1)

{

t1.terminalSet.append('@');

this->first('@');

}

s->right = new rightNode('@');

s->right->next=NULL;

q->next = s;

q = s;

}

else

{

s->right = new rightNode(findLeftT[k][0]);

rightNode \*s1 = s->right;

for(int k1 =1; k1<findLeftT[k].size();k1++)

{

rightNode \*q1 = new rightNode(findLeftT[k][k1]);

s1->next = q1;

s1 = q1;

}

s1->next=NULL;

q->next = s;

q =s;

}

}

q->next=NULL;

this->first(upperCase[0]); //求这个新符号的firstSet

upperCase.removeFirst(); //在大写字母中删除已用的非终结符号

}

else

{

bool flag = true; //用于判断文法是否出现在其他状态集合tempStatus[tempLeft[k]]里

for(int k =0;k<tempLeft.size();k++)

{

if(k!=j)

{

if(tempStatus[tempLeft[k]].indexOf(tempStatus[tempLeft[j]][0]) != -1 && tempStatus[tempLeft[k]].size()>1)

{

flag = false;

break;

}

}

}

if(flag)

{

save[tempStatus[tempLeft[j]][0]] = false; //不含公共左公因子的文法不删除

continue;

}

else

{

int n =0;

p = t1.textG;

QVector<QChar> temp;

while(p!=NULL)

{

if(p->leftCh == ch)

{

if(n == tempStatus[tempLeft[j]][0])

{

rightNode \*p1 = p->right;

temp = findTextRule(p1->rightCh,tempLeft[j],false,ch);

if(temp.empty())

{

QMessageBox::warning(this,"警告","输出文法非上下文无关文法！");

exit(-1);

}

if(temp[0]=='@')

temp.clear();

p1 = p1->next;

while(p1!=NULL)

{

temp.append(p1->rightCh);

p1 = p1->next;

}

break;

}

n++;

}

p = p->next;

}

while(p->next != NULL)

{

p = p->next;

}

textNode \*s = new textNode(ch);

if(temp.empty())

{

if(t1.terminalSet.indexOf('@')==-1)

{

t1.terminalSet.append('@');

this->first('@');

}

s->right = new rightNode('@');

s->right->next=NULL;

p->next = s;

p = s;

}

else

{

s->right = new rightNode(temp[0]);

rightNode \*s1 = s->right;

for(int k =1; k<temp.size();k++)

{

rightNode \*q1 = new rightNode(temp[k]);

s1->next = q1;

s1 = q1;

}

s1->next =NULL;

p->next = s;

p = s;

}

p->next=NULL;

}

}

}

p = t1.textG;

int n = 0;

textNode \*q = p;

while (p!=NULL) {

if(p->leftCh==ch)

{

if(save[n])

{

if(q == p)

{

q = p->next;

delete p;

p = q;

t1.textG = p;

}

else

{

q->next = p->next;

delete p;

p = q->next;

}

n++;

if(n==i)

break;

continue;

}

n++;

if(n==i)

break;

}

q = p;

p = p->next;

}

judgeLeft[ch] = 1;

delete []save;

}

void MainWindow::**leftFactor**() //提取左公因子

{

//先求一遍非终结符号的first集合，以便查看右部出现第一个字符为非终结符号时是否需要替换

this->first(t1.start);

for(int i=0;i<t1.non\_terminalSet.size();i++)

{

if(!firstMap.contains(t1.non\_terminalSet[i]))

this->first(t1.non\_terminalSet[i]);

}

for(int i=0;i<t1.terminalSet.size();i++)

this->first(t1.terminalSet[i]);

for(int i=0;i<t1.non\_terminalSet.size();i++)

judgeLeft[t1.non\_terminalSet[i]] = 0;

for(int i=0;i<t1.non\_terminalSet.size();i++)

{

if(judgeLeft[t1.non\_terminalSet[i]])

continue;

else

this->leftFactorChar(t1.non\_terminalSet[i]);

}

//输出存储结构中的化简后文法规则

textNode \*p =t1.textG;

QString str = "";

while(p!=NULL)

{

rightNode \*p1 = p->right;

str = str + p->leftCh + "->";

while(p1!= NULL)

{

str = str + p1->rightCh;

p1= p1->next;

}

str = str + "\n";

p = p->next;

}

ui->textEdit\_3->setText(str);

qDebug()<<t1.start;

str = "";

str = str + t1.start;

ui->lineEdit\_start2->setText(str);

str = "";

for(int i=0;i<t1.non\_terminalSet.size();i++)

{

str = str +t1.non\_terminalSet[i];

}

ui->lineEdit\_non2->setText(str);

str = "";

for(int i=0;i<t1.terminalSet.size();i++)

{

str = str +t1.terminalSet[i];

}

ui->lineEdit\_te2->setText(str);

}

### 7. 求first集合

QVector<QChar> MainWindow::**first**(QChar ch)

{

QVector<QChar> temp;

textNode \*p = t1.textG; //获取文法规则

firstMap[ch] = temp;

if(t1.terminalSet.indexOf(ch)!=-1) //若为终结符则first为自己，返回

{

firstMap[ch].append(ch);

return firstMap[ch];

}

while(p!=NULL)

{

if(p->leftCh == ch)

{

temp.clear();

rightNode \*p1 = p->right;

while(p1!=NULL)

{

if(t1.terminalSet.indexOf(p1->rightCh)!=-1)//若为终结符号，直接加入，否则求first

{

temp.append(p1->rightCh);

break;

}

else

{

QVector<QChar> temp1;

if(firstMap.contains(p1->rightCh))

temp1 = firstMap[p1->rightCh];

else

temp1 = first(p1->rightCh);

QVector<QChar> temp2 = temp1;

temp2.removeAll('@');

temp = this->vectors\_set\_union(temp,temp2);

if(temp1.indexOf('@') == -1)

break;

p1 = p1->next;

}

}

if(p1 == NULL)

temp.append('@');

firstMap[ch] = this->vectors\_set\_union(firstMap[ch],temp);

}

p = p->next;

}

return firstMap[ch];

}

### 8.求follow集合

void MainWindow::**follow**()

{

while(true)

{

tempMap = followMap;

textNode \*p = t1.textG;

while(p!=NULL)

{

rightNode \*p1 = p->right;

while(p1!=NULL)

{

if(t1.non\_terminalSet.indexOf(p1->rightCh)!=-1)

{

rightNode \*q1 = p1->next;

if(q1==NULL)

{

followMap[p1->rightCh] = this->vectors\_set\_union(followMap[p1->rightCh],followMap[p->leftCh]);

}

else

{

while(q1!=NULL)

{

if(t1.terminalSet.indexOf(q1->rightCh)!=-1)

{

if(followMap[p1->rightCh].indexOf(q1->rightCh) == -1)

followMap[p1->rightCh].append(q1->rightCh);

break;

}

else

{

QVector<QChar> temp = firstMap[q1->rightCh];

if(temp.indexOf('@')==-1)

{

followMap[p1->rightCh] = this->vectors\_set\_union(followMap[p1->rightCh],temp);

break;

}

else

{

temp.removeAll('@');

followMap[p1->rightCh] = this->vectors\_set\_union(followMap[p1->rightCh],temp);

q1 = q1->next;

}

}

}

if(q1 == NULL)

{

followMap[p1->rightCh] = this->vectors\_set\_union(followMap[p1->rightCh],followMap[p->leftCh]);

}

}

}

p1 = p1->next;

}

p = p->next;

}

if(followMap == tempMap)

break;

}

}

### 9.实现最左推导分析

void MainWindow::**left\_infer**() //最左推导

{

ui->tableWidget\_2->setRowCount(0);

ui->tableWidget\_2->setColumnCount(3); //三列，分析栈，输入，动作

QStringList VStrList;

VStrList.push\_back("分析栈");

VStrList.push\_back("输入");

VStrList.push\_back("动作");

ui->tableWidget\_2->setHorizontalHeaderLabels(VStrList);

QStack<QChar> s; //存放分析栈

QString sentence = ui->lineEdit->text(); //获取用户输入的句子

sentence = sentence;

QString sCh = ""; //存放分析栈的数据

s.push('$'); //终止符号入栈

s.push(t1.start); //文法开始符号入栈

sCh.append('$');

sCh.append(t1.start);

while(!s.empty())

{

QChar ch = s.pop(); //取出栈顶符号

if(ch == sentence[0])

{

if(sentence[0] == '$')

{

int count = ui->tableWidget\_2->rowCount();

ui->tableWidget\_2->insertRow(count);

ui->tableWidget\_2->setItem(count,0,new QTableWidgetItem(sCh));

ui->tableWidget\_2->setItem(count,1,new QTableWidgetItem(sentence));

ui->tableWidget\_2->setItem(count,2,new QTableWidgetItem("接受"));

sCh = sCh.left(sCh.size()-1); //取出栈顶符号

break;

}

else

{

int count = ui->tableWidget\_2->rowCount();

ui->tableWidget\_2->insertRow(count);

ui->tableWidget\_2->setItem(count,0,new QTableWidgetItem(sCh));

ui->tableWidget\_2->setItem(count,1,new QTableWidgetItem(sentence));

ui->tableWidget\_2->setItem(count,2,new QTableWidgetItem("匹配"));

sCh = sCh.left(sCh.size()-1); //取出栈顶符号

sentence.remove(0,1);

continue;

}

}

else

{

if(firstMap[ch].indexOf(sentence[0]) != -1)

{

if(followMap[ch].indexOf(sentence[0])!=-1 && firstMap[ch].indexOf('@')!=-1 && sentence[1] == "$")

{

int count = ui->tableWidget\_2->rowCount();

QString t1 = "";

t1 = t1 + ch + "->" + "@";

ui->tableWidget\_2->insertRow(count);

ui->tableWidget\_2->setItem(count,0,new QTableWidgetItem(sCh));

ui->tableWidget\_2->setItem(count,1,new QTableWidgetItem(sentence));

ui->tableWidget\_2->setItem(count,2,new QTableWidgetItem(t1));

sCh = sCh.left(sCh.size()-1); //取出栈顶符号

continue;

}

textNode \*p=t1.textG;

while(p!=NULL)

{

if(p->leftCh == ch)

{

rightNode \*p1=p->right;

if(firstMap[p1->rightCh].indexOf(sentence[0]) != -1)

{

int count = ui->tableWidget\_2->rowCount();

QString rule = "";

while(p1!=NULL)

{

rule = rule + p1->rightCh;

p1 = p1->next;

}

for(int i = rule.size()-1;i>=0;i--)

{

s.push(rule[i]);

}

QString t1 = "";

t1 = t1 + ch + "->" + rule;

ui->tableWidget\_2->insertRow(count);

ui->tableWidget\_2->setItem(count,0,new QTableWidgetItem(sCh));

ui->tableWidget\_2->setItem(count,1,new QTableWidgetItem(sentence));

ui->tableWidget\_2->setItem(count,2,new QTableWidgetItem(t1));

sCh = sCh.left(sCh.size()-1); //取出栈顶符号

for(int i = rule.size()-1;i>=0;i--)

{

sCh = sCh +rule[i];

}

break;

}

}

p = p->next;

}

}

else

{

if(followMap[ch].indexOf(sentence[0])!=-1)

{

if(firstMap[ch].indexOf('@')==-1)

{

QString t1 = "";

t1 = t1 + "此刻输入" +sentence[0] + "出错！";

int count = ui->tableWidget\_2->rowCount();

ui->tableWidget\_2->insertRow(count);

ui->tableWidget\_2->setItem(count,0,new QTableWidgetItem(sCh));

ui->tableWidget\_2->setItem(count,1,new QTableWidgetItem(sentence));

ui->tableWidget\_2->setItem(count,2,new QTableWidgetItem(t1));

break;

}

int count = ui->tableWidget\_2->rowCount();

QString t1 = "";

t1 = t1 + ch + "->" + "@";

ui->tableWidget\_2->insertRow(count);

ui->tableWidget\_2->setItem(count,0,new QTableWidgetItem(sCh));

ui->tableWidget\_2->setItem(count,1,new QTableWidgetItem(sentence));

ui->tableWidget\_2->setItem(count,2,new QTableWidgetItem(t1));

sCh = sCh.left(sCh.size()-1); //取出栈顶符号

}

else

{

QString t1 = "";

t1 = t1 + "此刻输入" +sentence[0] + "出错！";

int count = ui->tableWidget\_2->rowCount();

ui->tableWidget\_2->insertRow(count);

ui->tableWidget\_2->setItem(count,0,new QTableWidgetItem(sCh));

ui->tableWidget\_2->setItem(count,1,new QTableWidgetItem(sentence));

ui->tableWidget\_2->setItem(count,2,new QTableWidgetItem(t1));

break;

}

}

}

}

}

### 10.退出系统

//退出系统

connect(ui->pB\_exit,&QPushButton::clicked,this,[=](){

exit(0);

});