#include<bits/stdc++.h>

using namespace std;

string M[4]={"p|q","p|~q","~p|q","~p|~q"};

string m[4]={"p&q","p&~q","~p&q","~p&~q"};

string mat[4]={"M0","M1","M2","M3"};

int flag[4]={0};

string mat1[4]={"m0","m1","m2","m3"};

int Conditional(char str[]);

int DeMorgan(char str[]);

int DeleBracket(char str[]);

int DeleSame(char str[]);

bool subDeleSame(char );

string turnString(char str[]);

string absorbChar(char str1[]);

void matchM(string str);

string match\_m(string str);

int main() {

char str[10000]={"((p^q)&q)^p"};

Conditional(str);

cout<<str<<endl;

DeMorgan(str);

cout<<str<<endl;//输出蕴含处理结果

string s=absorbChar(str);

cout<<s<<endl;

cout<<endl;

matchM(s);

for(int i=0;i<4;i++){

if(flag[i]==1) cout<<mat[i]<<endl;

}

for(int i=0;i<4;i++){

if(flag[i]==0){

cout<<mat1[i];

if(i!=3) cout<<"|";

}

}

cout<<endl;

// for(int i=0;i<4;i++){

// if(flag[i]==1) cout<<M[i]<<endl;

// }

for(int i=0;i<4;i++){

if(flag[i]==0){

cout<<"(";

cout<<m[i];

cout<<")";

if(i!=3) cout<<"|";

}

}

return 0;

}

int Conditional(char str[]){

stack<char> stk;//处理前端 的stack

int bi/\*储存左指针\*/,ei;

char ans[10000];

for(int i=0; i<strlen(str); i++) { //遍历预处理字符

char fin[1000];//处理前端

char fin2[1000];

if(str[i]=='^') {

fin[0]='~';//在字符前加~和括号（

fin[1]='(';

int l=i,r=i;//l为左指针 r为右指针

//遇到括号

if(str[l-1]==')') {

int count=2;//作为fin的下标 fin[0]为~，fin[1]为（

l--;

int bracketNum=0;

while(str[l]!='('||bracketNum!=1) {//因为第一个')'要入stack所以 =1

if (str[l]==')')bracketNum++;

if(str[l]=='(')bracketNum--;//检查括号匹配

stk.push(str[l]);//把左括号之前的入栈

//cout<<stk.top()<<endl;

l--;//左移

}

bi=l;//储存左指针

while(!stk.empty()) {

fin[count++]=stk.top();

stk.pop();

}

} //无括号

else {

fin[1]=str[l-1];

bi=l-1;

}// 推的右边有括号

if(str[r+1]=='(') {

r++;

int count=0;

while(str[r]!=')') {//存在括号匹配问题

fin2[count++]=str[r];

r++;

}

ei=r;

fin2[count]=')';

} //右边无括号

else {

fin2[0]=str[r+1];

ei=r+1;

}

queue<char> que;//用于最后整合

for(int j=0; j<bi; j++)//将bi指针前面未处理部分入队列

que.push(str[j]);

for(int j=0; j<strlen(fin); j++)//将处理好的部分fin(运算符‘^’左边的运算数）入队列

que.push(fin[j]);

que.push('|');//插入‘|’运算符

for(int j=0; j<strlen(fin2); j++)//将'^' 右边的运算数入队列

que.push(fin2[j]);

for(int j=ei+1; j<strlen(str); j++)//将ei指针后面未处理部分入队列

que.push(str[j]);

int s=que.size();

for(int j=0; j<s; j++) {//推出所有队列元素附加给str

str[j]=que.front();

que.pop();

}

}

}

return 0;

}

int DeMorgan(char str[]){

int bi/\*储存左指针\*/,ei/\*储存右指针\*/;

int len=strlen(str);//获取长度

char str2[10000];

char a[]="";

queue<char> que;

char ans[10000];

strcpy(str2,str);//复制预处理字符储存到str2

for(int i=0;i<len;i++){

char fin[1000];

if(str2[i]=='~' && str2[i+1]=='('){//遇到~（

bi=i-1;//左指针

int j=i+2,cnt=0;//

int bracketNum=0;// 检查括号匹配

while(str2[j]!=')'||bracketNum!=0/\*j=j+2 '('不包含\*/){//没有遇到右括号

if (str[j]=='(')bracketNum++;

if(str[j]==')')bracketNum--;//检查括号匹配

if(str2[j]>='a' && str2[j]<='z') {

fin[cnt++]='~';

fin[cnt++]=str[j];//在字母前面加'~'

}

else if(str2[j]=='|') fin[cnt++]='&';

else if(str2[j]=='&')fin[cnt++]='|';

else fin[cnt++]=str2[j];//'~'照抄

j++;//指针向后扫

}

ei=j+1;//ei指针后面的未处理的部分

for(int j=0;j<=bi;j++)//bi前面的未处理的部分入队列

que.push(str2[j]);

for(int j=0;j<strlen(fin);j++)//将处理好的部分入队列

que.push(fin[j]);

for(int j=ei;j<len;j++){//ei指针后面的未处理的部分

que.push(str2[j]);

}

strcpy(str2,a);

int qLen=que.size();

for(int j=0;j<qLen;j++){

// cout<<que.front();

ans[j]=que.front();

que.pop();

}

// cout<<endl;

cout<<ans<<endl;

strcpy(str2,a);

strcpy(str2,ans);

}

}

int len1=strlen(ans);

for(int i=0;i<len1;i++){

if(ans[i]=='~' && ans[i+1]=='~'){

len1=len1-2;

for(int j=i;j<len1;j++) {

ans[j]=ans[j+2];

}

}

}

ans[len1]='\0';//加上结尾标识符 /0

//cout<<ans<<endl;

strcpy(str,ans);

}

string turnString(char str[]){

string ans;

for(int i=0;i<strlen(str);i++)

ans+=str[i];

return ans;

}

string absorbChar(char str1[]){

int begini,endi;

string str,ans="";

str=turnString(str1);

stack<char> stk[27];

queue<char> que;

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

if(str[i]=='('){

begini=i;

i++;

while(str[i]!=')'){

if(str[i]>='a' && str[i]<='z'){

stk[str[i]-'a'].push(str[i]);

stk[str[i]-'a'].push(str[i+1]);

}

i++;

endi=i;

}

}

if(str[i]>='a' && str[i]<='z'){

if(!stk[str[i]-'a'].empty()){

if((stk[str[i]-'a'].top()=='&' && str[i-1]=='|')|| (stk[str[i]-'a'].top()=='|' && str[i-1]=='&') ){

for(int j=begini;j<=endi;j++) str[j]=' ';

str[endi]=str[i];

str[i]=' ';

str[i-1]=' ';

}

}

}

}

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

if(str[i]!=' ') {

ans+=str[i];

}

}

// cout<<ans<<endl;

return ans;

}

int DeleBracket(char str[]){

char str2[10000];

strcpy(str2,str);

int len=strlen(str2);

for(int i=0;i<len;i++){

if(str2[i]=='(' || str2[i]==')'){

len--;

for(int j=i;j<len;j++) {

str2[j]=str2[j+1];

}

}

}

str2[len]='\0';

strcpy(str,str2);

cout<<str<<endl;

}

int DeleSame(char str[]){//p&~r&~p|r

string sTemp;

for(int i=0;i<strlen(str);i++)

sTemp+=str[i];

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

cout<<i<<": ";

char sameChar='0';//相同的运算数 初始化为零

bool sameCharFlag=false;

char sameOperator='0';//相同的运算符 初始化为零

if(sTemp[i]>='a'&&sTemp[i]<='z'){

sameChar=sTemp[i];

sameOperator= sTemp[i+1];//储存运算符和下一个运算数

int sameI=i;//储存下标

if(sTemp[i-1]=='~')

sameCharFlag=true;//前面有'~'

for(int j=i;j<sTemp.size();j++){

//如果遇到与前面运算符不同的运算运算符搜索停止

if(sTemp[j]!=sameOperator&&subDeleSame(sTemp[j]))

break;

if(sTemp[j]==sameChar){

int NumLeft=3;//左运算符删除数量

int NumRight=2;//右 运算符删除数量

int sameJ=j-1;

//if();

if(sameCharFlag==true){

if(sTemp[j-1]!='~'){//说明为~p&p型

//sameJ=j-1;

}

else{//说明为~p&~p型

NumRight++;//delet &~p

sameJ--;//sameJ=j-2;

}

sTemp.erase(sameI-1,NumLeft);

cout<<sTemp<<endl;

j-=NumLeft;

sTemp.erase(sameJ,NumRight);

//j-=NumRight;

}

else if(sameCharFlag==false){

int sameJ;

if(sTemp[j-1]=='~'){//p&~p

NumLeft--;

NumRight++;

sameJ--;//sameJ=j-2;

}

else{//p&p

}

sTemp.erase(sameI,NumLeft);//2

//cout<<sTemp<<endl;

j-=NumLeft;

sTemp.erase(sameJ,NumRight);//3

}

}

}

}

}

cout<<sTemp<<endl;

}

bool subDeleSame(char s){

switch(s){

case '^':case '&':case '|':

return true;

default:

return false;

}

}

void matchM(string str){

for(int i=0;i<4;i++){

if(str==M[i]){

flag[i]=1;

}

}

}

string match\_m(string str){

}

