注意:本篇文章篇幅较长，如果你只是想要代码而不关心具体实现的话，请移步至最后。

要求:输入任意一个命题公式，都能输出其真值表

(为使编程简单化，将离散数学中的逻辑运算符的实现改一下)

逻辑运算符:

取反:¬ 改为:!和！(前一个为英文状态下，后一个为中文状态下)

等价↔ 改为:=

合取∧ 改为:&

析取∨ 改为:|

蕴涵→ 改为:-

注意:在c++中，没有与蕴涵相对应的运算符，所以我们在运算时要用蕴涵等值律进行转化。

其中的难点:

1.除了以上的逻辑运算符之外，还得考虑括号的问题(括号优先级最高)

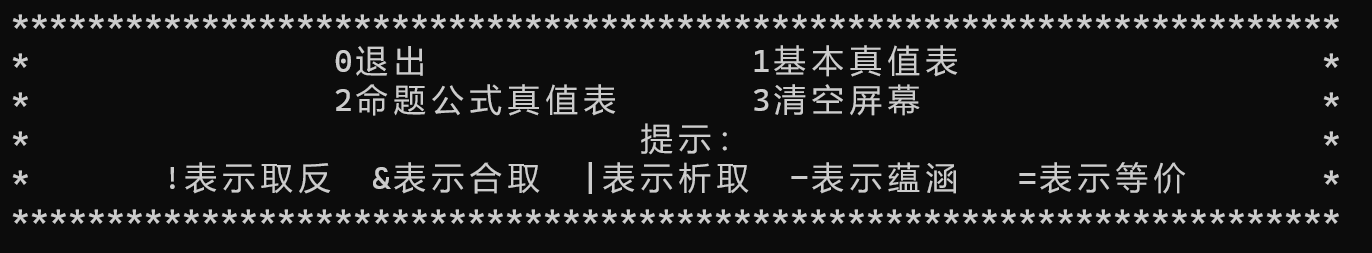
2.由于命题公式是随机给的，所以要先判断其是否是一个命题公式。

3.给定的命题公式中的命题符号数量不定，所以还要找出命题符号数，并给每一个命题符号赋值

4.计算

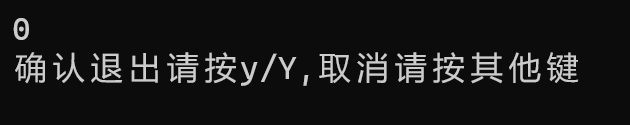
先来看看最终的结果:

主界面:(主要是不想让界面空荡荡的，并且提示输入的时候运算符被我们重新定义了)

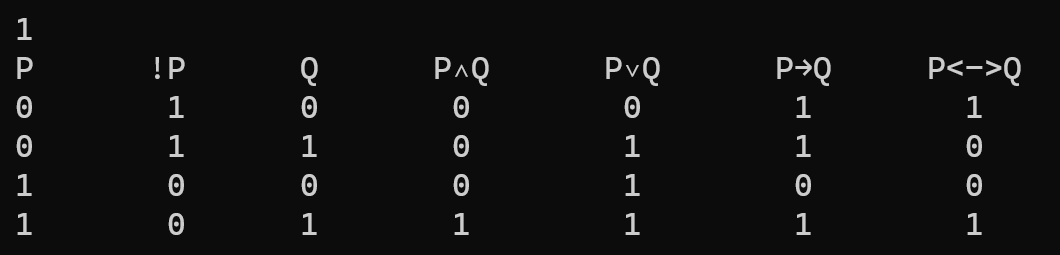


在主界面中，只有输入了相应的数字才能进行其操作(让你的程序看起来高大上一点)

操作0:



操作1:



操作2:



操作3就没必要展示了，输入其他的结果将会提示你重新输入

接下来就将这些用c++实现

首先先包含头文件

#include<iostream>

#include<string>

using namespace std;

然后将我们要写的函数的函数声明写出来:

//检查并将SIGN中没有的命题符号添加在SIGN后面

void voluation(string bb);

//输出基本真值表

void a();

//对当前的排列存储在pail数组中

void pint();

//判断当前的排列情况在pail数组中能否找到

int fun();

//对命题符号的值进行排列。如果在pail数组中没有找到该排列，就将该排列存储在pail数组中

void pai(int k);

//判断括号类型为((...))还是()...()

bool fn(string a,int j);

//取反运算时判断取反运算符后面还有没有取反运算符，再返回计算结果

int fal( string b,int i);

//查找命题符号在SIGN中的位置，找到返回其位置，否则返回-1

int ss(string a, int i);

//判断是字符'1'或者'0'，返回其对应的int类型，如果不是则返回-1

int mm(string a, int i);

//对命题公式进行计算

char a\_deal(string am);

接下来先把操作1实现:

void a() {

char n = 'P';

char m = 'Q';

int r;

cout << n << " " << "!" << n << " " << m;

cout << " " << n << "∧" << m ;

cout <<" " << n << "∨" << m;

cout <<" " << n << "→" << m;

cout <<" " << n << "<->" << m<< endl;

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

r = !i;

cout << i<< " " << r << " " << j;

r = i && j;

cout << " " << r;

r = i || j;

cout <<" " << r ;

r = !i || j;

cout <<" " << r ;

r = i == j;

cout <<" " << r <<endl;

}

}

}

在这里就只用了两个for循环就可以实现真值表的打印。

接下来就开始实现操作2:

首先要想实现操作2，我们得要先做一些准备工作

//命题符号最多数量，多一点是为了防止越界访问

const int SUM = 26;

//存储每个命题符号的不同赋值情况的全排列

int pail[129][SUM];

//存储当前的命题符号赋值情况

int A[SUM];

//存储命题符号

string SIGN = "0PQRST";

//控制命题符号的个数

int C = 0;

//控制命题符号赋值的全排列

int N = 1;

//控制当前正在使用的排列

int K = 1;

//取反运算符的数量

int NUM = 0;

class logic {

public:

string am;//用来存储命题

logic();

//检查am是不是一个正确的命题公式,正确返回true，错误返回false

bool a\_voluation();

//对命题符号及其命题进行输出

void a\_print()

{

for (int i = 1; i <=C; i++)

{

cout << SIGN[i] << "\t";

}

cout << this->am << endl;

}

};

//构造函数，让将命题初始化为字符0

logic::logic()

{

am = '0';

}

接下来就来实现比较复杂的函数，判断输入的式子是否是命题，如果是则对其进行计算。

//检查输入的命题公式是否正确

bool logic::a\_voluation()

{

int t = 0;

int n = 0;

int m = 0;

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

{

if (this->am[i] >= 'A' && this->am[i] <= 'Z'

|| this->am[i] >= 'a' && this->am[i] <= 'z'

|| this->am[i] == '1' && this->am[i] == '0')

{

n++; break;

}

}

if (n == 0)return false;

while (1)

{

int i=0;

for (i = 0; i < am.length(); i++)

{

if (this->am[i] == ' ')

{

m++;

for (int j = i; j < am.length()-1; j++)

{

am[j] = am[j + 1];

}

am.erase(am.length()-1, 1);

}

}

if (m == 0)break;

}

for (int i = 0; i < this->am.length(); i++)

{

if (this->am[i] >= 'A' && this->am[i] <= 'Z'

|| this->am[i] >= 'a' && this->am[i] <= 'z'

|| this->am[i] == '1' && this->am[i] == '0')

{

if (this->am[i+1] >= 'A' && this->am[i+1] <= 'Z'

||this->am[i+1] >= 'a' && this->am[i+1] <= 'z'

||this->am[i + 1] == '1' && this->am[i + 1] == '0'

||this->am[i + 1] == '('

|| this->am[i + 1] == '（'

|| this->am[i - 1] == ')'

|| this->am[i - 1] == '）')

{

return false;

}

}

if ((this->am[i] == '&'

|| this->am[i] == '|'

|| this->am[i] == '-'

|| this->am[i] == '=')&&i==0)

{

return false;

}

else if (this->am[i] == '&'

||this->am[i] == '|'

||this->am[i] == '-'

||this->am[i] == '='

||this->am[i] == '!'

||this->am[i] == '！')

{

if (i == am.length() - 1)return false;

if (this->am[i + 1] == '&'

|| this->am[i+1] == '|'

|| this->am[i+1] == '-'

|| this->am[i+1] == '=')

{

return false;

}

if ((this->am[i] == '&'

|| this->am[i] == '|'

|| this->am[i] == '-'

|| this->am[i] == '=')

&& (this->am[i + 1] == '!'

|| this->am[i + 1] == '！')

&& (this->am[i + 2] == '!'

|| this->am[i + 2] == '！')) {

return false;

}

}

else if (this->am[i] == '('|| this->am[i + 1] == '（')

{

t++;

if (this->am[i + 1] == '&'

|| this->am[i + 1] == '|'

|| this->am[i + 1] == '-'

|| this->am[i + 1] == '='

|| this->am[i + 1] == '）'

|| this->am[i + 1] == ')')return false;

}

else if (this->am[i] == ')' || this->am[i + 1] == '）')

{

t--;

if (this->am[i-1] == '&'

|| this->am[i-1] == '|'

|| this->am[i-1] == '-'

|| this->am[i-1] == '='

|| this->am[i-1] == '!'

|| this->am[i-1] == '！'

|| this->am[i + 1] == '!'

|| this->am[i + 1] == '！'

|| this->am[i + 1] == '('

|| this->am[i +1] == '（')return false;

}

if (t < 0)

{

return false;

}

}

if (t !=0)

{

return false;

}

return true;

}

其中判断命题是否错误主要是以下几点:

1.命题符号后面不能是命题符号

2.除了取反运算符之外，其他的运算符后面不能是非取反运算符

3.取反运算符不能出现在其他运算符前面

4.括号必须匹配，且不能出现")"先出现在"("前面

5."("前一个不能是命题符号或")"，后一个不能是运算符，")"后一个不能是命题符号、！或"("，前一个不能是运算符

6.每个命题至少有一个命题符号

判断正确后，先不着急计算，要先把命题符号获取并存储

//检查并将SIGN中没有的命题符号添加在SIGN后面

void voluation(string bb)

{

int i, j, t = 1;

for (i = 0; i < bb.length(); i++)

{

if (bb[i] >= 'A' && bb[i] <= 'Z' || bb[i] >= 'a' && bb[i] <= 'z')

{

for (j = 1; j <= C; j++)

{

if (SIGN[j] == bb[i])

{

break;

}

}

if (j == C + 1)

{

SIGN[t] = bb[i];

C++;

t++;

}

}

}

}

然后再对其命题进行全排列

//将当前的排列存储在pail数组中

void pint()

{

for (int i = 1; i <= C; i++)

{

pail[N][i] = A[i];

}

N++;

}

//判断当前的排列情况在不在pail数组中能否找到

int fun()

{

int t = 0;

for (int j = 1; j < N; j++)

{

t = 0;

for (int i = 1; i <= C; i++)

{

if (pail[j][i] == A[i])

{

t++;

if (t == C)

return 0;

}

}

}

return 1;

}

//对命题符号的值进行排列，如果在pail函数中没有找到该排列，就将该排列存储在pail数组中

void pai(int k)

{

int l = fun();

if (k >= C)

{

if (l == 1)

{

pint();

}

}

else

{

for (int i = k; i <= C; i++)

{

swap(A[k], A[i]);

pai(k + 1);

swap(A[k], A[i]);

}

}

}

在计算的时候，以运算符为准，找到其前一个命题符号和后一个命题符号在排列好的数组中的位置

接下来就实现查找函数

在SIGN数组中查找，找到返回相应的位置，没找到则返回-1

int ss(string a, int i)

{

for (int j = 0; j < SIGN.length(); j++)

{

if (a[i] == SIGN[j])

return j;

}

return -1;

}

如果运算过程中运用ss函数没找到，就用mm函数来查找运算对象是否为0或1

int mm(string a, int i)

{

if (a[i] == '1')return 1;

if (a[i] == '0')return 0;

else return -1;

}

查找函数实现之后就要进行计算，但在计算之前还要进行括号位置的判断，如果括号是(((……)))的和()……()运算不一样， 要将括号匹配好之后，将括号里面的内容用递归的算法来计算并返回计算结果。

bool fn(string a,int j)

{

for (int i = j; i < a.length(); i++)

{

if (a[i] == '(' || a[i] == '（')

{

return true;

}

if (a[i] == ')' || a[i] == '）')

{

return false;

}

}

return false;

}

对于取反这一特殊的运算符，其可以在取反后再取反，所以我们也要把取反运算符的数量数出来，然后只需要看奇偶数就可以进行计算

int fal(string b, int i)

{

int j = 0;

int m = i;

if (b[m - 1] == '!') {

j++; NUM++;

}

while (1)

{

if (b[m] == '!')j++;

if (b[m] != '!')break;

m++;

NUM++;

}

int t;

if (mm(b, m) != -1)

{

t = mm(b, m);

if (j % 2 == 0)return t;

else return !t;

}

else

{

t = ss(b, m);

if (j % 2 == 0)return pail[K][t];

else return !pail[K][t];

}

}

接下来就开始真正的计算了

char a\_deal(string am)

{

int a, b;

int temp = -1;

while (1)

{

a = 0, b = 0;

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

{

if (am[i] == '(' || am[i] == '（')

{

a = i;

if (fn(am, i+1))

{

for (int j = am.length() - 1; j > i; j--)

{

if (am[j] == ')' || am[j] == '）')

{

b = j;

break;

}

}

}

else

{

for (int j = i; j < am.length(); j++)

{

if (am[j] == ')' || am[j] == '）')

{

b = j;

break;

}

}

}

string bm;

int d = 1;

for (int k = a + 1; k < b; k++)

{

bm.push\_back(am[k]);

}

am[a] = a\_deal(bm);

if (b == am.length() - 1)

{

am.erase(a+1, b - a);

}

else

{

for (int n = b + 1; n < am.length(); n++)

{

am[a + d] = am[n];

d++;

}

am.erase(am.length() - b + a, b - a);

}

}

}

if (a == b&&a==0)break;

}

int i = 0;

int t1, t2;

while (1)

{

temp = -1;

for (i = 0; i < am.length(); i++)

{

switch (am[i])

{

case '!':

case '！':

t1 = fal(am, i + 1);

temp = t1;

i += NUM;

NUM = 0;

break;

case '&':

t1 = fal(am, i + 1);

if (temp == -1)

{

if (mm(am, i - 1) != -1)temp = t1 && mm(am, i - 1);

else { t2 = ss(am, i - 1); temp = t1 && pail[K][t2]; }

}

else temp = t1 && temp;

i += NUM;

NUM = 0;

break;

case '|':t1 = fal(am, i + 1);

if (temp == -1)

{

if (mm(am, i - 1) != -1)temp = t1 || mm(am, i - 1);

else { t2 = ss(am, i - 1); temp = t1 || pail[K][t2]; }

}

else temp = t1 || temp;

i += NUM;

NUM = 0;

break;

case '-':t1 = fal(am, i + 1);

if (temp == -1)

{

if (mm(am, i - 1) != -1)temp = !t1 || mm(am, i - 1);

else {

t2 = ss(am, i - 1);

temp = ((!pail[K][t2]) || t1);

}

}

else temp = (!temp) || t1;

i += NUM;

NUM = 0;

break;

case '=':t1 = fal(am, i + 1);

if (temp == -1)

{

if (mm(am, i - 1) != -1)temp = t1 == mm(am, i - 1);

else {

t2 = ss(am, i - 1);

temp = t1 == pail[K][t2];

}

}

else temp = t1 == temp;

i += NUM;

NUM = 0;

break;

default:

break;

}

}

if (temp == -1)

{

for (int l = 0; l < am.length(); l++)

{

int t = ss(am, l);

if (t != -1)

{

return pail[K][t]+'0';

}

else if (mm(am,l))

{

return mm(am, l)+'0';

}

}

}

else {

return temp + '0';

}

}

}

计算过程主要是如果有括号，先将括号问题进行匹配，然后将括号内的式子用递归的算法再次调用该函数。

如果没有括号，则以运算符为准，(temp为一个变量)如果temp=-1，则用temp来存储之前计算得出的值，然后再用这个值与下一个命题符号或0或1进行计算，最终返回temp的值，如果没有运算符，就只传入了一个命题符号，则在SIGN里面查找其位置并返回其值。

最后再来写一个菜单界面和输入函数

void pro()

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\* 0退出 1基本真值表 \*" << endl;

cout << "\* 2命题公式真值表 3清空屏幕 \*" << endl;

cout << "\* 提示： \*" << endl;

cout << "\* !表示取反 &表示合取 |表示析取 -表示蕴涵 =表示等价 \*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

void start(string& p)

{

cout << "请输入一个正确的命题公式，并按下Enter键" << endl;

cin >> p;

}

剩下的就只需要在main函数中进行调用就行了

int main()

{

while (1)

{

for (int i = 1; i <= C; i++)

{

A[i] = 0;

}

C = 0;

N = 1;

K = 1;

NUM = 0;

pro();

logic p;

char am;

cin >> am;

switch (am)

{

case '0':

char y;

cout << "确认退出请按y/Y,取消请按其他键" << endl;

cin >> y;

if (y == 'y' || y == 'Y')

{

exit(0);

}

break;

case '1':

a();

break;

case '2':

start(p.am);

if (p.a\_voluation())

{

voluation(p.am);

if (C > 7)

{

cout << "您输入的命题中含有的命题符号超过7，将不能计算" << endl;

break;

}

for (int w = 0; w <= C; w++)

{

pai(1);

A[w] = 1;

}

pai(1);

p.a\_print();

while (1)

{

for (int k = 1; k <= C; k++)

{

cout << pail[K][k] << "\t";

}

for (int mmam = 0; mmam < p.am.length() / 2; mmam++)

{

cout << " ";

}

cout << a\_deal(p.am)<<endl;

K++;

if (K >= N)break;

}

}

else

{

cout << "您输入了一个错误的命题" << endl;

};

break;

case '3':

system("cls");

break;

default:

cout << "输入错误，请重新输入"<<endl;

break;

}

}

return 0;

}

注意，在计算完成后需要将所有全局变量全部回归初始化状态，不然接下来再进行计算会出错

不过我并不建议像我这样写代码，最好是分一下文件写。我这代码并没有运用太过复杂的算法，所以运行速度上有些影响。

代码写得不好，还请多担待。

下面是打包好的代码和exe文件:

http://链接：https://pan.baidu.com/s/1Gew5fkFvULP\_1HBKdk-1WQ 提取码：1234