PROPOSITIONAL EQUIVALENCE:

If any two propositions are joined up by the phrase "if, and only if", the result is a compound proposition called an equivalence. The two propositions connected in this way are referred to as the left and right side of the equivalence. By asserting the equivalence of two propositions, it is intended to exclude the possibility that one is true and the other false; therefore, an equivalence is true if its left and right sides are either both true or both false, and otherwise the equivalence is false.

In other words, two propositions are called logically equivalent if, and only if, when same proposition  variables (or sentential variables) are used to represent identical compound propositions, their forms are logically equivalent.

**1. Tautology**

When a propositional expression is equivalent to TRUE, we call it a TAUTOLOGY:

*in a tautology, every possible possible way of assigning truth values to the propositions leads to an expression which is true.*

Q --> (~P v Q) <==> T

**2. Contradictions**

When a propositional expression is equivalent to FALSE, we call it a CONTRADICTION.

*in a contradiction, every possible possible way of assigning truth values to the propositions leads to an expression which is false.*

**3. Equivalence**

Expressions A, B are equivalent iff A <--> B is a tautology.

**4. Completeness of And, OR, NOT**

Can define an arbitrary propositional function, by specifying its truth values for a whole column of a truth table. We now constructively prove that we can express any function using just and, or, and not.

SOURCE CODE:

#include <bits/stdc++.h>

#define pb push\_back

using namespace std;

typedef unsigned int uint;

bool precedence(char o1, char o2) // returns true if o1 has the same precedence or higher precedence than o2

{

char op[] = { '!', '&', '|', 'c', 'b' };

uint i, j;

for(i = 0; i < 5; i++) if(op[i] == o1) break;

for(j = 0; j < 5; j++) if(op[j] == o2) break;

if(i <= j) return true;

return false;

}

string postfix(string q) // && = &, || = |, ! = !, <-> = b, -> = c

{

stack<char> op;

string post;

q = q + ')';

op.push('(');

uint s = q.size();

for(uint i = 0; !op.empty(); i++)

{

if(q[i] == 'T' || q[i] == 'F')

{

post += q[i];

}

else if(q[i] == '(')

{

op.push('(');

}

else if(q[i] == ')')

{

while(op.top() != '(')

{

post += op.top();

op.pop();

}

op.pop(); // pop the '('

}

else // operators

{

char o2 = q[i];

while(precedence(op.top(), o2))

{

post += op.top();

op.pop();

}

op.push(o2);

}

} // end of for

return post;

}

bool cond(const bool& a, const bool& b)

{

if(a && (!b)) return false;

return true;

}

bool bi(const bool& a, const bool& b)

{

if(a == b) return true;

return false;

}

bool solvePostfix(const string& post)

{

stack<bool> stk;

uint sz = post.size();

for(uint i = 0; i < sz; i++)

{

if(post[i] == 'T') stk.push(true);

else if(post[i] == 'F') stk.push(false);

else // operators

{

if(post[i] == '!') // unary

{

bool a = stk.top();

stk.pop();

// cout << !a << endl;

stk.push(!a);

}

else // binary

{

bool a = stk.top();

stk.pop();

bool b = stk.top();

stk.pop();

// b operator a

switch(post[i]) // && = &, || = |, ! = !, <-> = b, -> = c

{

case '&': stk.push(b && a); break;

case '|': stk.push(b || a); break;

case 'c': stk.push(cond(b, a)); break;

case 'b': stk.push(bi(b, a));

} // end of switch

} // end of else

} // end of else

} // end of for

return stk.top();

}

bool one[2][1] = { // p

true,

false

};

bool two[4][2] = { // p, q

true, true,

true, false,

false, true,

false, false

};

bool three[8][3] = { // p, q, r

true, true, true,

true, true, false,

true, false, false,

true, false, true,

false, true, true,

false, true, false,

false, false, false,

false, false, true

};

char truthValue(const bool& b) // bool to char

{

if(b) return 'T';

else return 'F';

}

bool equivalent\_1(string prop1, string prop2) // works with postfix notations

{

int i, j, k;

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

{

// traverse the first proposition and replace the value of p

for(k = 0; k < prop1.size(); k++)

{

if(prop1[k] == 'P') prop1[k] = truthValue( one[i][0] );

} // end of for

// traverse the second proposition

for(k = 0; k < prop2.size(); k++)

{

if(prop2[k] == 'P') prop2[k] = truthValue( one[i][1] );

} // end of for

prop1 = postfix(prop1);

prop2 = postfix(prop2);

if(solvePostfix(prop1) != solvePostfix(prop2)) return false;

} // end of for

return true;

}

bool equivalent\_2(const string& p1, const string& p2) // works with postfix notations

{

int i, j, k;

for(i = 0; i < 4; i++) // traverse row-wise

{

string prop1 = p1;

string prop2 = p2;

// traverse the first proposition and replace

for(k = 0; k < prop1.size(); k++)

{

if(prop1[k] == 'P') prop1[k] = truthValue( two[i][0] );

else if(prop1[k] == 'Q') prop1[k] = truthValue( two[i][1] );

} // end of for

// traverse the second proposition

for(k = 0; k < prop2.size(); k++)

{

if(prop2[k] == 'P') prop2[k] = truthValue( two[i][0] );

else if(prop2[k] == 'Q') prop2[k] = truthValue( two[i][1] );

} // end of for

prop1 = postfix(prop1);

prop2 = postfix(prop2);

if(solvePostfix(prop1) != solvePostfix(prop2)) return false;

} // end of for

return true;

}

bool equivalent\_3(const string& p1, const string& p2)

{

int i, j, k;

for(i = 0; i < 8; i++) // traverse row-wise

{

string prop1 = p1;

string prop2 = p2;

// traverse the first proposition and replace

for(k = 0; k < prop1.size(); k++)

{

if(prop1[k] == 'P') prop1[k] = truthValue( three[i][0] );

else if(prop1[k] == 'Q') prop1[k] = truthValue( three[i][1] );

else if(prop1[k] == 'R') prop1[k] = truthValue( three[i][3] );

} // end of for

// traverse the second proposition

for(k = 0; k < prop2.size(); k++)

{

if(prop2[k] == 'P') prop2[k] = truthValue( three[i][0] );

else if(prop2[k] == 'Q') prop2[k] = truthValue( three[i][1] );

else if(prop2[k] == 'R') prop2[k] = truthValue( three[i][3] );

} // end of for

prop1 = postfix(prop1);

prop2 = postfix(prop2);

if(solvePostfix(prop1) != solvePostfix(prop2)) return false;

} // end of for

return true;

}

int main()

{

int n;

string prop1, prop2;

cout<<"propositional equivalent checking\n";

cout<<"where, && = &, || = |, ! = !, <-> = b, -> = c\n";

cout<<"P, Q, R for variables\n";

printf("Enter number of variables: "); // && = &, || = |, ! = !, <-> = b, -> = c

scanf("%d", &n); // P, Q, R for variables

printf("Enter first proposition: ");

cin >> prop1;

printf("Enter second proposition: ");

cin >> prop2;

bool b;

switch(n)

{

case 1: b = equivalent\_1(prop1, prop2); break;

case 2: b = equivalent\_2(prop1, prop2); break;

case 3: b = equivalent\_3(prop1, prop2); break;

}

if(b) printf("Propositions are equivalent.\n");

else printf("Propositions are not equivalent.\n");

return 0;

}

OUTPUT:

