```
// BoolFuncPrj.cpp : Defines the entry point for the console application.
// Peter the Great St. Petersburg Polytechnic University,
// Institute of Applied Mathematics & Mechanics,
// Dep. Applied Mathematics, Serge V. Stakhov (c) 2018
/////////////////////////////////////////////////////////

#include "stdafx.h"
#include "BF.h"

int main()
{
    cout << "BoolFunc Project" << endl;
    BF ConstFalse;
    cout << "ConstFalse.Val(0) = " << ConstFalse.Val(0) << endl;
    // CTrue(true) calls ~BF():
    BF CFalse(false), CTrue(true);
    cout << "CFalse.Val(0) = " << CFalse.Val(0) << " CTrue.Val(0) = " << CTrue.Val ⤾
      (0) << endl;

    // TVT:
    uint  X // n-dim. bool vector packed
        , andTVT[1] = { 0x8 }   // &
        ,  orTVT[1] = { 0xE }   // |
        , impTVT[1] = { 0xD }   // =>
    ;
    BF and(2, andTVT), or(2, orTVT), imp(2, impTVT);
    cout << "BF(byte N, uint *p, RepT t = tTVT) objects have been built" << endl;
    // ZheP:
    uint  AndZheP[1] = { 0x8 } // AndZhePcoeffSupp[1] = {0x3}, // ZheP[&] = x0 &   ⤾
      x1;
    // OrZhePcoeffSupp[3]={0x1, 0x2, 0x3}, // ZheP[Or](x0, x1) = x0 ^ x1 ^ x0 & x1;⤾

    //(d(x0)=0x1, d(x1)=0x2, d(x1&x1)=0x3) -> IndOrZhePcoeffSupp=2+4+8=14;
        , OrZheP[1] = { 0xE } // {14}
        // ImpZhePcoeffSupp[3]={0x0, 0x1, 0x3},// ZheP[=>](x0, x1)=1 ^ x0 ^ x0 &   ⤾
          x1;
        //(d(1)=0x0, d(x0)=0x1, d(x1&x1)=0x3)->IndOrZhePcoeffSupp=1+2+8=11;
        , ImpZheP[1] = { 0xB }; // {11}
    BF And(2, AndZheP, tZheP), Or(2, OrZheP, tZheP), Imp(2, ImpZheP, tZheP);
    cout << "BF(N, p, tZheP) objects have been built" << endl;

/*  // Test and.Val(X) == And.Val(X, tZheP)? :
    bool andValXeqAndValXtZheP;
    for (andValXeqAndValXtZheP = true, X = 0; X < 4 && andValXeqAndValXtZheP; X++)
        andValXeqAndValXtZheP = and.Val(X) == And.Val(X, tZheP);
    if (andValXeqAndValXtZheP)
        cout << "for All X (and.Val(X) == And.Val(X, tZheP))" << endl;
    else {
        X--;
        cout << "Exists X = " << X
             << " (and.Val(X) == " << and.Val(X) << " != "
             << And.Val(X, tZheP) << " == And.Val(X, tZheP))" << endl;
    } // if (andValXeqAndValXtZheP)
    // Test or.Val(X) == Or.Val(X, tZheP)? :
    bool orValXeqOrValXtZheP;
    for (orValXeqOrValXtZheP = true, X = 0; X < 4 && orValXeqOrValXtZheP; X++)
```

```
        orValXeqOrValXtZheP = or.Val(X) == Or.Val(X, tZheP);
    if (orValXeqOrValXtZheP)
        cout << "for All X (or.Val(X) == Or.Val(X, tZheP))" << endl;
    else {
        X--;
        cout << "Exists X = " << X
            << " (or.Val(X) == " << or.Val(X) << " != "
            << Or.Val(X, tZheP) << " == Or.Val(X, tZheP))" << endl;
    } // if (orValXeqOrValXtZheP)
    // Test imp.Val(X) == Imp.Val(X, tZheP)? :
    bool impValXeqImpValXtZheP;
        for (impValXeqImpValXtZheP = true, X = 0; X < 4 && impValXeqImpValXtZheP; X
          ++)
        impValXeqImpValXtZheP = imp.Val(X) == Imp.Val(X, tZheP);
    if (impValXeqImpValXtZheP)
        cout << "for All X (imp.Val(X) == Imp.Val(X, tZheP))" << endl;
    else { X--; // after for( ; ; X++)
        cout << "Exists X = " << X
            << " (imp.Val(X) == " << imp .Val(X) << " != "
            << Imp.Val(X, tZheP) << " == Imp.Val(X, tZheP))" << endl;
    } // if (impValXeqImpValXtZheP)
*/ // ^ Test and.Val(X) == And.Val(X, tZheP)? ^
    // Test and.Val(X) == And.Val(X)? :
    bool andValXeqAndValX;
    for (andValXeqAndValX = true, X = 0; X < 4 && andValXeqAndValX; X++)
        andValXeqAndValX = and.Val(X) == And.Val(X);
    if (andValXeqAndValX)
        cout << "for All X (and.Val(X) == And.Val(X))" << endl;
    else {
        X--;
        cout << "Exists X = " << X
            << " (and.Val(X) == " << and.Val(X) << " != "
            << And.Val(X) << " == And.Val(X))" << endl;
    } // if (andValXeqAndValX)
    // Test or.Val(X) == Or.Val(X)? :
    bool orValXeqOrValX;
    for (orValXeqOrValX = true, X = 0; X < 4 && orValXeqOrValX; X++)
        orValXeqOrValX = or .Val(X) == Or.Val(X);
    if (orValXeqOrValX)
        cout << "for All X (or.Val(X) == Or.Val(X))" << endl;
    else {
        X--;
        cout << "Exists X = " << X
            << " (or.Val(X) == " << or .Val(X) << " != "
            << Or.Val(X) << " == Or.Val(X))" << endl;
    } // if (orValXeqOrValX)
    // Test imp.Val(X) == Imp.Val(X)? :
    bool impValXeqImpValX;
    for (impValXeqImpValX = true, X = 0; X < 4 && impValXeqImpValX; X++)
        impValXeqImpValX = imp.Val(X) == Imp.Val(X);
    if (impValXeqImpValX)
        cout << "for All X (imp.Val(X) == Imp.Val(X))" << endl;
    else {
        X--; // after for( ; ; X++)
        cout << "Exists X = " << X
            << " (imp.Val(X) == " << imp.Val(X) << " != "
            << Imp.Val(X) << " == Imp.Val(X))" << endl;
    } // if (impValXeqImpValX)

/*  // X is out of range [0 .. 2^n]:
    try {
        bool ConstFalseVal_1 = ConstFalse.Val(1);
        cout << "ConstFalse.Val(1) = " << ConstFalseVal_1 << endl;
    } // try
    catch (const std::exception& MyExc) {
        cout << "catch(MyExc): " << MyExc.what() << endl;
        return -1;
    } // catch
*/ // ^ X is out of range [0 .. 2^n] ^

/* Test `hex` I/O stream:
    uint AllUnits = 0xFFFFFFFF;
    cout << "AllUnits = " << AllUnits << endl;
    std::ios_base::fmtflags oldFlags;
    oldFlags = cout.setf(std::ios_base::hex, std::ios_base::basefield);
    cout << "AllUnits = " << AllUnits << endl;
    cout.setf(oldFlags);
    cout << "AllUnits = " << AllUnits << endl;
*/

/* Test Grey:
    const byte nMax = 10; // 5;
    byte n; // number of bool variables
    uint Ind, G, Gind, GrGind;
    bool GreyOK;

    cout.setf(ios::hex);
    // Check: Grey2Int(Grey(Ind)) == Ind
    for (n = 0, GreyOK = true; n <= nMax && GreyOK; n++) {
        for (Ind = 0; Ind < 1U << n && GreyOK; Ind++) {
            G = Grey(Ind);
            Gind = Grey2Int(G);
        //  cout << " Ind = " << Ind << " G = " << G << " Gind = " << Gind << endl;
            GreyOK = Gind == Ind;
        } // for (Ind = 0; Ind < 1 << n && GreyOK; Ind++)
        cout << "n = " << int(n) << ": (Grey2Int(Grey(Ind)) == Ind) = " <<
          (GreyOK ? "true" : "false") << endl;
    } // for (n = 0, GreyOK = true; n <= nMax && GreyOK; n++)
    cout << "-----------------------------------------" << endl;
    // Check: Grey(Grey2Int(G)) == G
    for (n = 0, GreyOK = true; n <= nMax && GreyOK; n++) {
        for (G = 0; G < 1U << n && GreyOK; G++) {
        //  G = Grey(Ind);
            Gind = Grey2Int(G);
            GrGind = Grey(Gind);
            //  cout << " G = " << G << " Gind = " << Gind << " GrGind = " <<
              GrGind << endl;
            GreyOK = GrGind == G;
        } // for (Ind = 0; Ind < 1 << n && GreyOK; Ind++)
        cout << "n = " << int(n) << ": Grey(Grey2Int(G)) == G = " << (GreyOK ?
          "true" : "false") << endl;
    } // for (n = 0, GreyOK = true; n <= nMax && GreyOK; n++)
```

```
*/ // ^ Test Grey ^
    return 0;
}
```