```
// "BF.cpp"
// 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"

// Grey Code of Index
uint Grey(uint Ind) { return Ind ^ Ind >> 1; }

// Index of Grey Code
uint Grey2Int(uint G) {
    uint d, b, m;//Most Significant Bit xOR-Accumulator, bit-mask
    for (m = 1U << 31; ~(G & m) & m; m >>= 1); // G Most Significant Bit
    for (d = b = 0; m; b ^= G & m, d ^= b, b >>= 1, m >>= 1); //!
    return d; // Index of G
} // Grey2Int

BF::BF() { //const false == OR(NULL) == AND(OR(NULL)) == +(NULL)
    n = 0;
    mTVT = 1, TVT = new uint[mTVT], TVT[0] = 0x0; // == false
    mPDNF = 0, PDNF = new uint[mPDNF]; // OR(NULL) == false
    mPCNF = 1, PCNF = new uint[mPCNF], PCNF[0] = 0x0; // == NULL
    mZheP = 0, ZheP = new uint[mZheP]; // +(NULL) == false
    mKarn = 1, LKarn = 1, Karn = new uint*[mKarn],
        Karn[0] = new uint[LKarn], Karn[0][0] = 0x0; // == false
    mRDNF = 0, RDNF = new uint*[mRDNF]; // OR(NULL) == false
    mDF = 0, DF = new uint*[mDF]; // OR(NULL) == false
    mCF = 1, CF = new uint*[mCF], CF[0] = new uint[2],
        CF[0][0] = CF[0][1] = 0x0;//==NULL; AND(OR(NULL))==false
} // BF::BF()

BF::BF(bool TV) : BF() { // const BF == TrueValue
    if (TV) { // const true == OR(AND(NULL))==AND(NULL) == +(true)
        this -> ~BF(); // destroy (*this) == ConstFalse & rebuild:
        n = 0;
        mTVT = 1, TVT = new uint[mTVT], TVT[0] = 0x1; // == true
        mPDNF = 1, PDNF = new uint[mPDNF], PDNF[0] = 0x0; //NULL
        mPCNF = 0, PCNF = new uint[mPCNF];  // AND(NULL) == true
        mZheP = 1, ZheP = new uint[mZheP], ZheP[0] = 0x1; //true
        mKarn = 1, LKarn = 1, Karn = new uint*[mKarn],
            Karn[0] = new uint[LKarn], Karn[0][0] = 0x1;//==true
        mRDNF = 1, RDNF = new uint*[mRDNF], RDNF[0] = 0x0;//NULL
        mDF = 1, DF = new uint*[mDF], DF[0] = new uint[2],
            DF[0][0] = DF[0][1] = 0x0;//==NULL; OR(AND(NULL))==true
        mCF = 0, CF = new uint*[mCF]; // AND(NULL) == true
    } // if (TV)
} // BF::BF(bool TV)

/*
byte n;
size_t mTVT, mPDNF, mPCNF, mZheP, mKarn, LKarn, mRDNF, mDF, mCF;
uint *TVT, *PDNF, *PCNF, *ZheP, **Karn, **RDNF, **DF, **CF;
*/
```

```
// PDNF, PCNF, ZheP -> TVT -> PDNF, PCNF, ZheP, Karn, RDNF:
BF::BF(byte N, uint *p, RepT t, size_t M) {
    if (32 <= N) {
        cout << "BF(byte N,..): 32 <= N = " << int(N) << endl;
        throw NoutOfRange;
    } // if (32 <= N)
    n = N; // number of boolean variables < 32
    size_t i; uint X, TVTLbit = 1U << n;
    switch (t) {
    case tTVT:
        mTVT = n < 5 ? 1U : 1U << (n-5); //num. of 32-bit records
        try {
            TVT = new uint[mTVT];
        }
        catch (bad_alloc) // (const std::exception&)
        {
            cout << "BF(byte N, uint *p, RepT t): t = " << t
                 << " N = " << int(N) << "mTVT = " << mTVT
                 << " bad_alloc" << endl;
            throw BFnewArrayFail;
        } // try | catch
        for (i = 0; i < mTVT; i++) TVT[i] = p[i];
        break; // tTVT
    case tPDNF:
        mPDNF = M;
        try {
            PDNF = new uint[mPDNF];
        }
        catch (bad_alloc) // (const std::exception&)
        {
            cout << "BF(byte N, uint *p, RepT t): t = " << t
                << " N = " << int(N) << "mPDNF = " << mPDNF
                << " bad_alloc" << endl;
            throw BFnewArrayFail;
        } // try | catch
        for (i = 0; i < mPDNF; i++) PDNF[i] = p[i];
        break; // tPDNF
    case tPCNF:
        mPCNF = M;
        try {
            PCNF = new uint[mPCNF];
        }
        catch (bad_alloc) // (const std::exception&)
        {
            cout << "BF(byte N, uint *p, RepT t): t = " << t
                << " N = " << int(N) << "mPCNF = " << mPCNF
                << " bad_alloc" << endl;
            throw BFnewArrayFail;
        } // try | catch
        for (i = 0; i < mPCNF; i++) PCNF[i] = p[i];
        break; // tPCNF
    case tZheP:
        mZheP = n < 5 ? 1U : 1U << (n - 5); //num. of 32-bit records
        try {
            ZheP = new uint[mZheP];
        }
```

```cpp
        catch (bad_alloc) // (const std::exception&)
        {
            cout << "BF(byte N, uint *p, RepT t): t = " << t
                << " N = " << int(N) << "mZheP = " << mZheP
                << " bad_alloc" << endl;
            throw BFnewArrayFail;
        } // try | catch
        for (i = 0; i < mZheP; i++) ZheP[i] = p[i];
        break; // tZheP
    default:
        cout << "BF(byte, uint *, RepT t): UnKnown t = " << t << endl;
        throw IllegalRepTinBF;
    } // switch (t)
    if (t != tTVT) {
        mTVT = n < 5 ? 1U : 1U << (n - 5); //num. of 32-bit records
        try {
            TVT = new uint[mTVT];
        }
        catch (bad_alloc) // (const std::exception&)
        {
            cout << "BF(byte N, uint *p, RepT t): t = " << t
                << " N = " << int(N) << "mTVT = " << mTVT
                << " bad_alloc" << endl;
            throw BFnewArrayFail;
        } // try | catch
        for (X = 0; X < TVTLbit; X++)
            writeTVT(X, Val(X, t));
    } //
} // BF::BF(byte N, uint *p, RepT t = tTVT)

// Karn, DF, CF -> TVT -> PDNF, PCNF, ZheP, Karn, RDNF:
BF::BF(byte N, uint **p, RepT t, size_t M)
{
    if (32 <= N) {
        cout << "BF(byte N,..): 32 <= N = " << int(N) << endl;
        throw NoutOfRange;
    } // if (32 <= N)
    n = N; // number of boolean variables
//  size_t i;
    uint X, TVTLbit = 1U << n;
    switch (t) {
    case tKarn:
        break;
    case tRDNF:
        break;
    case tDF:
        break;
    case tCF:
        break;
    default:
        cout << "BF(byte, uint **, RepT t): UnKnown t = " << t << endl;
        throw IllegalRepTinBF;
    } // switch (t)
    for (X = 0; X < TVTLbit; X++)
        writeTVT(X, Val(X, t));
} // BF::BF(byte N, uint **p, RepT t = tKarn)
```

```cpp
bool BF::Val(uint X, RepT t) {
    uint D = 1U << n, Xmask = 0xFFFFFFFFU >> (32 - n);
    if (D <= X) {
        cout << "Val(X, t): n = " << int(n)
             << " X = " << X << " t = " << t << endl;
        throw XoutOfRange;
    } // (1U << n <= X)
    switch (t) {
    case tTVT:  return (TVT[X >> 5] >> (X & 0x1F) & 1U) == 1U;
    //          return (TVT[X >> 5] & 1U << (X & 0x1F)) != 0U;
    case tPDNF: return false;
    case tPCNF: return false;
    case tZheP: uint d, ZhePfX;
//      size_t wind, bind;  // for d=0..2^n-1:
//      wind = d >> 5;      // d / 32 == word's index
//      bind = X & 0x1F;    // d % 32 == bit's index
//      bool C[d]; // ZhePf binary Coefficient of multi-degree d
// ZheP[f](X) = ^(&(d[i] => X[i], i=0 .. n-1) & C[d]), d in bool^n);
        for (ZhePfX = d = 0; d < D; d++)
            ZhePfX ^=
            (   (ZheP[d >> 5] >> (d & 0x1F) & 1U) == 1U  // C[d] == 1
            && ((~d | X) & Xmask) == Xmask  // &(d[i] => x[i], i=0 .. n-1)
            ) ? 1U : 0;
        return ZhePfX == 1U;
    case tKarn: return false;
    case tRDNF: return false;
    case tDF:   return false;
    case tCF:   return false;
    default: cout << "Val(X, t): UnKnown RepT t = " << t << endl;
        throw UnKnownRepT;
    //  return false;
    } // switch (t)
} // bool BF::Val(uint X, RepT t = tTVT)

void BF::writeTVT(uint X, bool TVX) { // TVT(X) <- TVX:
    TVT[X >> 5] = TVT[X >> 5] ^ TVT[X >> 5] & 1U << (X & 0x1F) ^ (TVX ? 1U << (X & ⤶
      0x1F) : 0);
} // void BF::writeTVT(uint X)

BF::~BF() {
    size_t i;
    for (i = 0; i < mCF; i++) delete[] CF[i];
    delete[] CF;
    for (i = 0; i < mDF; i++) delete[] DF[i];
    delete[] DF;
    for (i = 0; i < mRDNF; i++) delete[] RDNF[i];
    delete[] RDNF;
    for (i = 0; i < mKarn; i++) delete[] Karn[i];
    delete[] Karn;
    delete[] ZheP;
    delete[] PCNF;
    delete[] PDNF;
    delete[] TVT;
} // BF::~BF()
```