Dominik Ciesiołkiewicz 44289 – Sprawozdanie Lab 8 Kodowanie kanałowe – TD

**Dla przejrzystości na końcu kodu umieściłem zrzuty z konsoli ze zwykłego kodowania jak i SECDEC*

Kod:

```
#include <iostream>
#include <fstream>
#include <complex>

using namespace std;

int lengthOfString(string str)
{
    return str.length();
}

string S2BS(string in, bool choice)  //String To Binary Stream
{
    string out = "";

    int n = in.length();
    string bity = "";

    if (choice == 1)//LittleEndian
    {
        for (int i = 0; i < n; i++)
        {
            int wartosc = int(in[i]);
            bity = "";

            while (wartosc > 0)
            {
                if (wartosc % 2)
                {
                    bity += '1';
                }
                else
                {
                    bity += '0';
                }
                wartosc = wartosc / 2;
            }
            out += bity;
        }
        reverse(out.begin(), out.end());
        //cout << out << endl;
        return out;
    }
    else {//BigEndian
        for (int i = 0; i < n; i++)
        {
            int wartosc = int(in[i]);
            bity = "";
```

```
            while (wartosc > 0)
            {
                if (wartosc % 2)
                {
                    bity += '1';
                }
                else
                {
                    bity += '0';
                }
                wartosc = wartosc / 2;
            }
            reverse(bity.begin(), bity.end());
            out += bity;
        }
        //cout << out << endl;
        return out;
    }
}

int* Hamming(string d)
{
    int G[7][4] = {
{1,1,0,1},{1,0,1,1},{1,0,0,0},{0,1,1,1},{0,1,0,0},{0,0,1,0},{0,0,0,1} };
    int* K = new int[7];

    for (int i = 0; i < 7; i++)
    {
        K[i] = 0;
        for (int j = 0; j < 4; j++)
        {
            //cout << G[i][j];
            //cout<<d[j]<<endl;
            K[i] += G[i][j] * (d[j] - '0');
        }
        //cout << endl;
    }

    cout << "K:" << endl;
    for (int i = 0; i < 7; i++)
    {
        cout << K[i] << endl;
    }
    cout << endl;

    cout << "K modulo 2:" << endl;
    for (int i = 0; i < 7; i++)
    {
        K[i] = K[i]%2;
        cout << K[i] << endl;
    }
    cout << endl;

    return K;
}

int* HammingSECDEC(string d)
{
    int G[7][4] = {
{1,1,0,1},{1,0,1,1},{1,0,0,0},{0,1,1,1},{0,1,0,0},{0,0,1,0},{0,0,0,1} };
    int* K = new int[8];
```

```
    for (int i = 0; i < 7; i++)
    {
        K[i] = 0;
        for (int j = 0; j < 4; j++)
        {
            K[i] += G[i][j] * (d[j] - '0');
        }
    }

    cout << "K:" << endl;
    for (int i = 0; i < 7; i++)
    {
        cout << K[i] << endl;
    }
    cout << endl;

    cout << "K modulo 2:" << endl;
    for (int i = 0; i < 7; i++)
    {
        K[i] = K[i] % 2;
        cout << K[i] << endl;
    }
    cout << endl;

    //Dla SECDEC:
    cout << "Ze sprawdzajacym bitem: " << endl;
    int err = 0;
    for (int i = 0; i < 7; i++)
    {
        err += K[i];
    }
    err = err % 2;
    K[7] = err;

    for (int i = 0; i < 8; i++)
    {
        cout << K[i] << endl;
    }
    cout << endl;

    return K;
}

int* DecHamming(int * K)
{
    {
        int H[3][7] = { {1,0,1,0,1,0,1},{0,1,1,0,0,1,1},{0,0,0,1,1,1,1} };
        int* KD = new int[7];

        for (int i = 0; i < 3; i++)
        {
            KD[i] = 0;
            for (int j = 0; j < 7; j++)
            {
                //cout << K[j];
                KD[i] += H[i][j] * K[j];
                //cout << KD[i];
            }
            //cout << endl;
        }

        cout << "K zdekodowane:" << endl;
```

```
        for (int i = 0; i < 3; i++)
        {
            cout << KD[i] << endl;
        }
        cout << endl;

        cout << "K zdekodowane modulo 2:" << endl;
        for (int i = 0; i < 3; i++)
        {
            KD[i] = KD[i] % 2;
            cout << KD[i] << endl;
        }
        cout << endl;

        return KD;
    }
}

int* DecHammingSECDEC(int* K)
{
    int H[3][7] = { {1,0,1,0,1,0,1},{0,1,1,0,0,1,1},{0,0,0,1,1,1,1} };
    int* KD = new int[7];

    cout << "Sprawdzanie p4:" << endl;
    int err = 0;
    for (int i = 0; i < 7; i++)
    {
        err += K[i];
    }
    err = err % 2;

    if (err != K[7])
    {
        cout << "P4 nie jest zgodne. Mamy 50% szans na powodzenie naprawy." << endl <<
endl;
    }
    else
    {
        cout << "P4 jest zgodne" << endl << endl;
    }

    int p1 = (K[0] + K[2] + K[4] + K[6])%2;
    int p2 = (K[1] + K[2] + K[5] + K[6])%2;
    int p3 = (K[3] + K[4] + K[5] + K[6])%2;

    int n = p1 * 1 + p2 * 2 + p3 * 4 - 1;

    cout << "Poprawiony kod odebrany:" << endl;
    if (K[n] == 0)
    {
        K[n] = 1;
    }
    else
    {
        K[n] = 0;
    }

    for (int i = 0; i < 8; i++)
    {
        cout << K[i] << endl;
    }
```

```
    cout << endl << "Sprawdzanie p4 - ponowne:" << endl;
    n = 0;
    for (int i = 0; i < 7; i++)
    {
        n += K[i];
    }
    n = n % 2;

    if (n != K[7])
    {
        cout << "P4 nie jest zgodne. Sa co najmniej 2 bledne bity. Odrzucamy pakiet."
<< endl << endl;
        return NULL;
    }
    else
    {
        cout << "P4 jest zgodne, odkodowujemy:" << endl << endl;

        cout << "Informacja odkodowana:" << endl;
        cout << K[2] << endl;
        cout << K[4] << endl;
        cout << K[5] << endl;
        cout << K[6] << endl;
    }

    return K;
}

int* BitNegation(int* K, int NoBit)
{
    if (K[NoBit] == 0)
        K[NoBit] = 1;
    else
        K[NoBit] = 0;
    return K;
}

int main()
{
    bool SECDEC = 0;// 0-zwykly kod Hamminga; 1-SECDEC

    string str = S2BS("8", 1);
    int n = lengthOfString(str);//4?

    cout << "Informacja:" << endl;
    // 1 1 1 0
    for (int i = 0; i < 4; i++)
    {
        cout << str[i] << endl;
    }
    cout << endl;

    if (SECDEC == 0)
    {
        int* K = Hamming(str);
        K = BitNegation(K, 2);
        //K: 2 2 1 2 1 1 0
        //K%2: 0 0 1 0 1 1 0

        cout << "K po negacji bitu 2:" << endl;
        //K po negacji bitu 2: 0 0 0 0 1 1 0
        for (int i = 0; i < 7; i++)
```

```
    {
        cout << K[i] << endl;
    }
    cout << endl;

    //K zdekodowane: 1 1 2
    //K zdekodowane modulo 2: 1 1 0

    int* D = DecHamming(K);

    if (D[0] == 0 && D[1] == 0 && D[2] == 0)
    {
        cout << "Kod nie posiada bledu. Transmisja poprawna." << endl << endl;
    }
    else
    {
        cout << "Transmisja zawiera blad na bicie nr:" << endl;
        //2
        int err = (D[0] + D[1]*2 + D[2]*4)-1;
        cout << err << endl << endl;

        cout << "Poprawiony kod odebrany:" << endl;
        //0 0 1 0 1 1 0
        if (K[err] == 0)
        {
            K[err] = 1;
        }
        else
        {
            K[err] = 0;
        }

        for (int i = 0; i < 7; i++)
        {
            cout << K[i] << endl;
        }
        cout << endl;
    }

    cout << "Informacja odkodowana:" << endl;
    // 1 1 1 0
    cout << K[2] << endl;
    cout << K[4] << endl;
    cout << K[5] << endl;
    cout << K[6] << endl;
}
else
{
    int* K = HammingSECDEC(str);

    K = BitNegation(K, 2);
    //K = BitNegation(K, 4);

    //K: 2 2 1 2 1 1 0
    //K%2: 0 0 1 0 1 1 0
    //K z bitem spr: 0 0 1 0 1 1 0 1

    cout << "K po negacji bitu 2:" << endl;
    //K po negacji bitu 2: 0 0 0 0 1 1 0 1
    for (int i = 0; i < 8; i++)
    {
        cout << K[i] << endl;
```

```
        }
        cout << endl;

        int* D = DecHammingSECDEC(K);
        //Poprawiony kod odebrany: 0 0 1 0 1 1 0 1
        //Informacja odkodowana: 1 1 1 0
    }
    return 1;
}
```

Zrzuty z konsoli:

Wersja zwykła:

```
Transmisja zawiera blad na bicie nr:
2

Poprawiony kod odebrany:
0
0
1
0
1
1
0

Informacja odkodowana:
1
1
1
0
```

Wersja SECDEC:

```
Konsola debugowania programu Microsoft Visual Studio
Informacja:
1
1
1
0

K:
2
2
1
2
1
1
0

K modulo 2:
0
0
1
0
1
1
0

Ze sprawdzajacym bitem:
0
0
1
0
1
1
0
1

K po negacji bitu 2:
0
0
0
0
1
1
0
1
```

```
Sprawdzanie p4:
P4 nie jest zgodne. Mamy 50% szans na powodzenie naprawy.

Poprawiony kod odebrany:
0
0
1
0
1
1
0
1

Sprawdzanie p4 - ponowne:
P4 jest zgodne, odkodowujemy:

Informacja odkodowana:
1
1
1
0
```