

# Kodowanie - projekt

Hubert Drożdżiak

## Spis treści

<b>1</b>	<b>Środowisko programistyczne</b>	<b>2</b>
<b>2</b>	<b>Zasada działania programu</b>	<b>2</b>
<b>3</b>	<b>Opis i analiza kodu źródłowego</b>	<b>2</b>
3.1	dodawanie_modulo()	2
3.2	wstep()	3
3.3	main()	3
3.3.1	Wstępne zarezerwowanie pamięci	3
3.3.2	Wprowadzanie danych do programu	3
3.3.3	Sprawdzenie poprawności podanego wielomianu generacyjnego	4
3.3.4	Wygenerowanie i wypisanie słów informacyjnych	5
3.3.5	Wytworzenie macierzy generującej niesystematycznej	5
3.3.6	Usystematyzowanie macierzy generującej	6
3.3.7	Wypisanie macierzy generującej do konsoli	6
3.3.8	Wytworzenie i wypisanie do konsoli macierzy zabezpieczającej	7
3.3.9	Wytworzenie i wypisanie do konsoli wszystkich słów kodowych	7
3.3.10	Obliczenie i wypisanie długości minimalnej, zdolności detekcyjnej i korekcyjnej	8
3.3.11	Pętla sprawdzająca czy podane słowo kodowe należy do tablicy słów kodowych (za pomocą wyliczonego syndromu)	9
<b>4</b>	<b>Instrukcja obsługi programu</b>	<b>11</b>
<b>5</b>	<b>Przykład użycia programu dla n=17, k=8, wielomian generujący: 1101001011</b>	<b>11</b>
5.1	Wpisanie danych oraz otrzymana tablica słów informacyjnych	11
5.2	Macierz generująca i macierz zabezpieczająca	12
5.3	Tablica słów kodowych	13
5.4	Odległość minimalna, zdolność detekcyjna i korekcyjna	13
5.5	Przykład dla dekodowania słowa kodowego 11111000010100001 wpisanego raz poprawnie, a raz z dwoma błędami	14

# 1 Środowisko programistyczne

Do napisania programu zostało użyte darmowe środowisko Code::Blocks.

## 2 Zasada działania programu

Po włączeniu aplikacji użytkownik wpisuje długość słowa kodowego (n), długość słowa informacyjnego (k) oraz wielomian generacyjny w postaci binarnej.

Po wpisaniu tych danych, program:

- sprawdza poprawność wpisanego wielomianu generacyjnego (metodą dzielenia wielomianów),
- wyświetla wszystkie słowa informacyjne,
- wyświetla macierz generującą systematyczną,
- wyświetla macierz zabezpieczającą,
- wyświetla wszystkie słowa kodowe,
- oblicza i wyświetla odległość minimalną, zdolność detekcyjną i zdolność korekcyjną.

Następnie użytkownik wpisuje słowo kodowe, a program sprawdza jego poprawność z użyciem obliczanego syndromu.

## 3 Opis i analiza kodu źródłowego

### 3.1 dodawanie\_modulo()

```
string dodawanie_modulo(string a, string b)
{
    if (a == b)
        return "0";
    return "1";
}

char dodawanie_modulo(char a, string b)
{
    const char *x = b.c_str();
    char *wsk = &a;
    if (*wsk == *x)
        return '0';
    return '1';
}
```

Funkcje zwracają wartość dodawania modulo 2.

## 3.2 wstep()

```
void wstep()
{
    for(int i=0; i<50; i++)
        cout << "#";
    cout << endl;
    cout << "#";
    for(int i=0; i<9; i++)
        cout << " ";
    cout << "APLIKACJA KODUJACO-DEKODUJACA";
    for(int i=0; i<10; i++)
        cout << " ";
    cout << "#" << endl;
    for(int i=0; i<50; i++)
        cout << "#";
    cout << endl;
}
```

Funkcja służy do wyświetlenia napisu początkowego.

## 3.3 main()

### 3.3.1 Wstępne zarezerwowanie pamięci

```
wstep();
char tablica_slow_informacyjnych[32768][15];
char tablica_slow_kodowych[32768][40];
string macierz_generujaca[70][70];
string macierz_zabezpieczajaca[100][50];
string odebrany_ciag[50];
string syndrom[50];
int n, k;
string wielomian_generujacy;
```

### 3.3.2 Wprowadzanie danych do programu

```
cout << "Podaj n (n<41): ";
do{
    cin >> n;
}while(n>40 || n<1);
cout << "Podaj k (k<16): ";
do{
    cin >> k;
}while(k>15 || k<1);
cout << "Podaj wielomian generujacy w postaci binarnej (CG2) (np. 101 dla x^2+1): ";
```

### 3.3.3 Sprawdzenie poprawności podanego wielomianu generacyjnego

```
int sumajedynek=0;
do{
    cin >> wielomian_generujacy;
    char wielomian[wielomian_generujacy.size()];

    for(int i=0; i<wielomian_generujacy.size(); i++)
        wielomian[i]=wielomian_generujacy[i];

    char dzielna[n+1];
    for(int i=0; i<n+1; i++)
    {
        if(i==0 || i==n)
            dzielna[i]='1';
        else
            dzielna[i]='0';
    }

    int poczatek = n;
    for(int q=n-wielomian_generujacy.size()+1; q>-1; q--)
    {
        if(dzielna[poczatek]=='1')
        {
            int iteracje=0;
            for(int i=0; i<wielomian_generujacy.size(); i++)
            {
                if(dzielna[poczatek-iteracje]==wielomian[i])
                    dzielna[poczatek-iteracje]='0';
                else
                    dzielna[poczatek-iteracje]='1';
                iteracje++;
            }
            poczatek--;
        }
    }

    sumajedynek=0;
    for(int a=n; a>-1; a--)
        if(dzielna[a]=='1')
            sumajedynek++;
    if(sumajedynek!=0)
    {
        cout << "Reszta z dzielenia wielomianu wynosi: ";
        for(int a=n; a>-1; a--)
            cout << dzielna[a];
        cout << endl;
        cout << "Podano bledny wielomian generacyjny, prosze wprowadzic jeszcze raz: ";
    }
}while(sumajedynek!=0);
```

### 3.3.4 Wygenerowanie i wypisanie słów informacyjnych

```
int ileLiczby_01 = pow(2, k - 1);
for (int kolumna = 0; kolumna < k; kolumna++)
{
    for (int rzad = 0; rzad < pow(2, k);)
    {
        for (int w = 0; w < ileLiczby_01; w++)
        {
            tablica_slow_informacyjnych[rzad][kolumna] = '0';
            rzad++;
        }
        for (int w = 0; w < ileLiczby_01; w++)
        {
            tablica_slow_informacyjnych[rzad][kolumna] = '1';
            rzad++;
        }
    }
    ileLiczby_01 = ileLiczby_01 / 2;
}

cout << "Słowa informacyjne: ";
for (int rzad = 0; rzad < pow(2, k); rzad++)
{
    for (int kolumna = 0; kolumna < k; kolumna++)
        cout << tablica_slow_informacyjnych[rzad][kolumna];
    if (rzad != pow(2, k) - 1)
        cout << ", ";
}
cout << endl;
```

### 3.3.5 Wytworzenie macierzy generującej niesystematycznej

```
for (int rzad = 0; rzad < k; rzad++)
{
    int NminusK = n - k;
    for (int kolumna = rzad; kolumna < 1 + n - k + rzad; kolumna++)
    {
        macierz_generujaca[rzad][n - kolumna - 1] = wielomian_generujacy[NminusK];
        NminusK--;
    }
    for (int kolumna = 0; kolumna < n; kolumna++)
    {
        if (macierz_generujaca[rzad][kolumna].length() == 0)
            macierz_generujaca[rzad][kolumna] = "0";
    }
}
```

### 3.3.6 Usystematyzowanie macierzy generującej

```
while (true)
{
    int ileZle = 0;
    for (int rzad = 0; rzad < k; rzad++)
    {
        for (int kolumna = 0; kolumna < k; kolumna++)
        {
            if (macierz_generujaca[rzad][kolumna] != "0")
                if (kolumna != rzad)
                    ileZle++;
        }
    }
    if (ileZle == 0)
        break;

    for(int rzadP=0; rzadP<k; rzadP++)
    {
        for(int kolumnaP=0; kolumnaP<k; kolumnaP++)
        {
            if(rzadP==kolumnaP)
                if(macierz_generujaca[rzadP][kolumnaP]!="1")
                {
                    for(int rzadX=0; rzadX<k; rzadX++){
                        if(macierz_generujaca[rzadX][kolumnaP]=="1"&&rzadX!=kolumnaP)
                            for(int i=0; i<n; i++)
                                macierz_generujaca[rzadP][i]=dodawanie_modulo(macierz_generujaca[rzadX][i],macierz_generujaca[rzadP][i]);
                    }
                }
        }
    }

    for(int rzadP=0; rzadP<k; rzadP++)
    {
        for(int kolumnaP=0; kolumnaP<k; kolumnaP++)
        {
            if(rzadP!=kolumnaP)
                if(macierz_generujaca[rzadP][kolumnaP]=="1")
                {
                    for(int rzadX=0; rzadX<k; rzadX++)
                        if(macierz_generujaca[rzadX][kolumnaP]=="1"&& rzadX!=rzadP)
                            for(int i=0; i<n; i++)
                                macierz_generujaca[rzadP][i]=dodawanie_modulo(macierz_generujaca[rzadX][i],macierz_generujaca[rzadP][i]);
                }
        }
    }
}
cout << endl;
```

### 3.3.7 Wypisanie macierzy generującej do konsoli

```
cout << "Macierz generujaca systematyczna" << endl;
for (int rzad = 0; rzad < k; rzad++)
{
    for (int kolumna = 0; kolumna < n; kolumna++)
        cout << macierz_generujaca[rzad][kolumna];
    cout << endl;
}
cout << endl;
```

### 3.3.8 Wytworzenie i wypisanie do konsoli macierzy zabezpieczającej

```
for (int rzadMG = 0; rzadMG < k; rzadMG++)
{
    int kolumnaMZ = 0;
    for (int kolumnaMG = k; kolumnaMG < n; kolumnaMG++)
    {
        macierz_zabezpieczajaca[rzadMG][kolumnaMZ] = macierz_generujaca[rzadMG][kolumnaMG];
        kolumnaMZ++;
    }
}
for (int rzad = k; rzad < n; rzad++)
{
    for (int kolumna = 0; kolumna < n - k; kolumna++)
    {
        if (kolumna + k == rzad)
            macierz_zabezpieczajaca[rzad][kolumna] = "1";
        else
            macierz_zabezpieczajaca[rzad][kolumna] = "0";
    }
}

cout << "Macierz zabezpieczajaca" << endl;
for(int i=0; i<n; i++){
    for(int j=0; j<n-k; j++)
        cout << macierz_zabezpieczajaca[i][j];
    cout << endl; } cout << endl;
```

### 3.3.9 Wytworzenie i wypisanie do konsoli wszystkich słów kodowych

```
for (int rzad = 0; rzad < pow(2, k); rzad++)
    for (int kolumna = 0; kolumna < n; kolumna++)
        tablica_slow_kodowych[rzad][kolumna] = '0';

for (int rzadTSK = 0; rzadTSK < pow(2, k); rzadTSK++)
{
    for (int kolumnaTSK = 0; kolumnaTSK < n; kolumnaTSK++)
    {
        for (int kolumnaTSI = 0; kolumnaTSI < k; kolumnaTSI++)
        {
            if (tablica_slow_informacyjnych[rzadTSK][kolumnaTSI] == '1')
                tablica_slow_kodowych[rzadTSK][kolumnaTSK] = dodawanie_modulo(tablica_slow_kodowych[rzadTSK][kolumnaTSK], macierz_generujaca[kolumnaTSI][kolumnaTSK]);
        }
    }
}

cout << "Tablica slow kodowych: ";
for(int i=0; i<pow(2,k); i++)
{
    for(int j=0; j<n; j++)
        cout << tablica_slow_kodowych[i][j];
    if(i!=pow(2,k)-1)
        cout << ", ";
}
```

### 3.3.10 Obliczenie i wypisanie długości minimalnej, zdolności detekcyjnej i korekcyjnej

```
int iloscJedynek=0;
for(int i=1; i<pow(2,k); i++)
{
    int ileJedynekTemp=0;
    for(int j=0; j<n; j++)
    {
        if(tablica_slow_kodowych[i][j]=='1')
            ileJedynekTemp++;
    }
    if(i==1)
        iloscJedynek=ileJedynekTemp;
    if(ileJedynekTemp<iloscJedynek)
        iloscJedynek=ileJedynekTemp;
}

int dmin = iloscJedynek;
int korekcja = (dmin-1)/2;

cout << endl; cout << endl;
cout << "Odleglosc minimalna kodu: " << dmin << endl; cout << "Zdolnosc detekcyjna kodu: " << dmin-1 << endl;
cout << "Zdolnosc korekcyjna kodu: " << korekcja << endl;

cout << endl;
```



### 3.3.11 Pętla sprawdzająca czy podane słowo kodowe należy do tablicy słów kodowych (za pomocą wyliczonego syndromu)

```
while(true){
    string odebrany_ciag_x;
    cout << "Podaj odebrany ciag (lub wpisz 'w' zeby zakonczyc): ";
    do{
        cin >> odebrany_ciag_x;

        if(odebrany_ciag_x=="w")
            return 0;
        if(odebrany_ciag_x.length()!=n)
        {
            cout << "Podano bledny ciag" << endl;
            cout << "Podaj odebrany ciag: ";
        }
    }while(odebrany_ciag_x.length()!=n);

    for(int i=0; i<n; i++)
        odebrany_ciag[i]=odebrany_ciag_x[i];

    for(int i=0; i<n-k; i++)
        syndrom[i]="0";

    for(int i=0; i<n-k; i++)
    {
        for(int j=0; j<n; j++)
        {
            if(odebrany_ciag[j]=="1")
                syndrom[i]=dodawanie_modulo(syndrom[i], macierz_zabezpieczajaca[j][i]);
        }
    }
    cout << "syndrom: ";
    for(int i=0; i<n-k; i++)
        cout << syndrom[i];
    cout << endl;
    int wagaH=0;
    for(int i=0; i<n-k; i++)
        if(syndrom[i]=="1")
            wagaH++;

    if(wagaH==0){
        cout << "Odebrano poprawny ciag" << endl;
        cout << "Uzyte slowo kodowe: ";
        for(int i=0; i<k; i++)
            cout << odebrany_ciag[i];
        cout << endl;
    }
    else
    {
        if(wagaH>0 && wagaH<=korekcja)
        {
            cout << "Podany ciag posiada blad w czesci zabezpieczajacej" << endl;
            cout << "Ciag po poprawieniu: ";
            int kolumnaA=0;
            for(int kolumnaX=k; kolumnaX<n; kolumnaX++)
            {
                odebrany_ciag[kolumnaX]=dodawanie_modulo(odebrany_ciag[kolumnaX], syndrom[kolumnaA]);
                kolumnaA++;
            }

            for(int i=0; i<n; i++)
                cout << odebrany_ciag[i];
            cout << endl;
            cout << "Zdekodowane slowo informacyjne: ";
            for(int i=0; i<k; i++)
                cout << odebrany_ciag[i];
            cout << endl;
        }
    }
}
```

```

else{
if(korekcja!=0){
int przestawienia=0;
for(int powt=0; powt<n; powt++){
{
przestawienia++;
string slowo0[l]=odebrany_ciag[0];
for(int i=0; i<n; i++){
{
if(i!=n-1)
odebrany_ciag[i]=odebrany_ciag[i+1];
else
odebrany_ciag[i]=slowo0[0];
}
for(int i=0; i<n-k; i++)
syndrom[i]="0";
for(int i=0; i<n-k; i++)
{
for(int j=0; j<n; j++){
{
if(odebrany_ciag[j]=="1")
syndrom[i]=dodawanie_modulo(syndrom[i], macierz_zabezpieczajaca[j][i]);
}
}
}
cout << "Syndrom po " << przestawienia << " przesuniecieciach: ";
for(int i=0; i<n-k; i++)
cout << syndrom[i];
cout << endl;

wagaH=0;
for(int i=0; i<n-k; i++)
if(syndrom[i]=="1")
wagaH++;
if(wagaH<=korekcja)
break;
}
if(wagaH<=korekcja)
{
cout << "Podany ciag posiada blad w czesci kodowej" << endl;
cout << "Odebrany ciag przesuniety " << przestawienia << " razy w lewo: ";

for(int i=0; i<n; i++)
cout << odebrany_ciag[i];
cout << endl;

cout << "Syndrom wynosi: ";
for(int i=0; i<n-k; i++)
cout << syndrom[i];
cout << endl;

cout << "Odebrany przesuniety ciag po poprawieniu: ";
int kolumnaA=0;
for(int kolumnaX=k; kolumnaX<n; kolumnaX++)
{
odebrany_ciag[kolumnaX]=dodawanie_modulo(odebrany_ciag[kolumnaX], syndrom[kolumnaA]);
kolumnaA++;
}
for(int i=0; i<n; i++)
cout << odebrany_ciag[i];
cout << endl;

for(int i=przestawienia; i>0; i--)
{
string pamiec[l]=odebrany_ciag[n-1];
for(int i=n-1; i>=1; i--){
{
if(i!=0)
{
odebrany_ciag[i]=odebrany_ciag[i-1];
}
else
{
odebrany_ciag[0]=pamiec[0];
}
}
}
}
cout << "Poprawiony odebrany ciag: ";
for(int i=0; i<n; i++)
cout << odebrany_ciag[i];
cout << endl;

cout << "Zdekodowane slowo informacyjne: ";
for(int i=0; i<k; i++)
cout << odebrany_ciag[i];
cout << endl;
}
}
}
if(wagaH>korekcja)
cout << "Odebrany ciag jest bledny i niekorygowalny" << endl; cout << endl;
}
}

```

## 4 Instrukcja obsługi programu

1. wpisać n
2. wpisać k
3. wpisać wielomian generujący w postaci binarnej
4. wpisywać kolejne słowa kodowe do sprawdzenia czy należą do danej tablicy słów kodowych

## 5 Przykład użycia programu dla $n=17$ , $k=8$ , wielomian generujący: 1101001011

### 5.1 Wpisanie danych oraz otrzymana tablica słów informacyjnych

```
C:\Users\huber\Desktop\Studia\kodowanie\Kodowanie-projekt\program.exe
#####
#          APLIKACJA KODUJACO-DEKODUJACA          #
#####
Podaj n (n<41): 17
Podaj k (k<16): 8
Podaj wielomian generujacy w postaci binarnej (CG2) (np. 101 dla  $x^2+1$ ): 1101001011
Słowa informacyjne: 00000000, 00000001, 00000010, 00000011, 00000100, 00000101, 00000110,
00000111, 00001000, 00001001, 00001010, 00001011, 00001100, 00001101, 00001110, 00001111,
00010000, 00010001, 00010010, 00010011, 00010100, 00010101, 00010110, 00010111, 00011000,
00011001, 00011010, 00011011, 00011100, 00011101, 00011110, 00011111, 00100000, 00100001,
00100010, 00100011, 00100100, 00100101, 00100110, 00100111, 00101000, 00101001, 00101010,
00101011, 00101100, 00101101, 00101110, 00101111, 00110000, 00110001, 00110010, 00110011,
00110100, 00110101, 00110110, 00110111, 00111000, 00111001, 00111010, 00111011, 00111100,
00111101, 00111110, 00111111, 01000000, 01000001, 01000010, 01000011, 01000100, 01000101,
01000110, 01000111, 01001000, 01001001, 01001010, 01001011, 01001100, 01001101, 01001110,
01001111, 01010000, 01010001, 01010010, 01010011, 01010100, 01010101, 01010110, 01010111,
01011000, 01011001, 01011010, 01011011, 01011100, 01011101, 01011110, 01011111, 01100000,
01100001, 01100010, 01100011, 01100100, 01100101, 01100110, 01100111, 01101000, 01101001,
01101010, 01101011, 01101100, 01101101, 01101110, 01101111, 01110000, 01110001, 01110010,
01110011, 01110100, 01110101, 01110110, 01110111, 01111000, 01111001, 01111010, 01111011,
01111100, 01111101, 01111110, 01111111, 10000000, 10000001, 10000010, 10000011, 10000100,
10000101, 10000110, 10000111, 10001000, 10001001, 10001010, 10001011, 10001100, 10001101,
10001110, 10001111, 10010000, 10010001, 10010010, 10010011, 10010100, 10010101, 10010110,
10010111, 10011000, 10011001, 10011010, 10011011, 10011100, 10011101, 10011110, 10011111,
10100000, 10100001, 10100010, 10100011, 10100100, 10100101, 10100110, 10100111, 10101000,
10101001, 10101010, 10101011, 10101100, 10101101, 10101110, 10101111, 10110000, 10110001,
10110010, 10110011, 10110100, 10110101, 10110110, 10110111, 10111000, 10111001, 10111010,
10111011, 10111100, 10111101, 10111110, 10111111, 11000000, 11000001, 11000010, 11000011,
11000100, 11000101, 11000110, 11000111, 11001000, 11001001, 11001010, 11001011, 11001100,
11001101, 11001110, 11001111, 11010000, 11010001, 11010010, 11010011, 11010100, 11010101,
11010110, 11010111, 11011000, 11011001, 11011010, 11011011, 11011100, 11011101, 11011110,
11011111, 11100000, 11100001, 11100010, 11100011, 11100100, 11100101, 11100110, 11100111,
```

## 5.2 Macierz generująca i macierz zabezpieczająca

```
C:\Users\huber\Desktop\Studia\kodowanie\Kodowanie-projekt\program.exe
11111010, 11111011, 11111100, 11111101, 11111110, 11111111

Macierz generujaca systematyczna
10000000110100101
01000000101110111
00100000100011110
0001000010001111
00001000111100010
00000100011110001
00000010111011101
00000001101001011

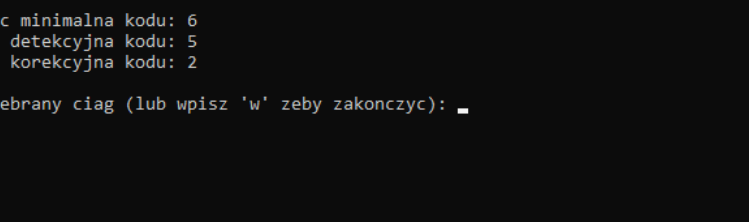
Macierz zabezpieczajaca
110100101
101110111
100011110
010001111
111100010
011110001
111011101
101001011
100000000
010000000
001000000
000100000
000010000
000001000
000000100
000000010
000000001

Tablica slow kodowych: 0000000000000000, 00000001101001011, 00000010111011101, 0000001101
```

### 5.3 Tablica słów kodowych

[illegible]

## 5.4 Odległość minimalna, zdolność detekcyjna i korekcyjna



WybierzC:\Users\huber\Desktop\Studia\kodowanie\Kodowanie-projekt\program.exe

Odleglosc minimalna kodu: 6  
Zdolnosc detekcyjna kodu: 5  
Zdolnosc korekcyjna kodu: 2

Podaj odebrany ciąg (lub wpisz 'w' zeby zakonczyc): \_

5.5 Przykład dla dekodowania słowa kodowego 11111000010100001 wpisanego raz poprawnie, a raz z dwoma błędami

```
C:\Users\huber\Desktop\Studia\kodowanie\Kodowanie-projekt\program.exe

Podaj odebrany ciąg (lub wpisz 'w' zeby zakonczyc): 11111000010100001
syndrom: 000000000
Odebrano poprawny ciąg
Uzyte slowo kodowe: 11111000
Podaj odebrany ciąg (lub wpisz 'w' zeby zakonczyc): 11011100010100001
syndrom: 111101111
Syndrom po 1 przesunięciach: 010010101
Syndrom po 2 przesunięciach: 100101010
Syndrom po 3 przesunięciach: 100011111
Syndrom po 4 przesunięciach: 101110101
Syndrom po 5 przesunięciach: 110100001
Syndrom po 6 przesunięciach: 000001001
Podany ciąg posiada błąd w części kodowej
Odebrany ciąg przesunięty 6 razy w lewo: 00010100001110111
Syndrom wynosi: 000001001
Odebrany przesunięty ciąg po poprawieniu: 00010100001111110
Poprawiony odebrany ciąg: 11111000010100001
Zdekodowane słowo informacyjne: 11111000

Podaj odebrany ciąg (lub wpisz 'w' zeby zakonczyc): _
```