Исходный текст программы Sword

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Text.RegularExpressions;
using System.Threading;
using System.Xml;
using Word = Microsoft.Office.Interop.Word;

namespace sword
{
    public partial class Form1 : Form
    {



        #region Настройка формы

        public Form1()
        {
            InitializeComponent();

            preset();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            setDefault();
        }

        //закрытие приложения
        private void Form1_FormClosed(object sender, FormClosedEventArgs e)
        {
            exitOfficeWord();
        }

        /// <summary>
        /// стройка формы
        /// </summary>
        private void setDefault()
        {
            this.Location = new Point((Screen.PrimaryScreen.Bounds.Width -
this.Width) / 2, (Screen.PrimaryScreen.Bounds.Height - this.Height) / 2);


            panelWord.Visible = false;

            setRBEncoding();

            currentEncoding = Encoding.Unicode;

            changeViewWordFace(false);
            changeArxivView(false);

            setEnableGroupColor(true);

            tempStart();
        }
```

```
        #endregion

        #region Смена режима работы

        //указывает включен расширенный режим или нет
        Boolean _advanceRegim;

        /// <summary>
        /// смена режима работы
        /// </summary>
        public Boolean advaceRegim
        {
            get { return _advanceRegim; }
            set
            {
                //изменить состояние режима
                _advanceRegim = value;

                //изменить отображение режима
                groupBoxAdvance.Visible = _advanceRegim;

                //при необходимости поменять текущий элемент в раскрывающемся
списке режимов работы
                if (_advanceRegim == true && toolStripComboBox1.SelectedIndex
== 0)
                {
                    toolStripComboBox1.SelectedIndex = 1;
                }
                else if (_advanceRegim == false &&
toolStripComboBox1.SelectedIndex == 1)
                {
                    toolStripComboBox1.SelectedIndex = 0;
                }
            }
        }

        //изменение режима работы (простой|расширенный) с помощью
раскрывающегося списка
        private void toolStripComboBox1_SelectedIndexChanged(object sender,
EventArgs e)
        {
            advaceRegim = (toolStripComboBox1.SelectedIndex == 1);
        }


        #endregion

        /// <summary>
        /// исходные данные для внедрения цвета
        /// </summary>
        private void setDefaultImplant()
        {
            currentEncoding = Encoding.GetEncoding(1251);
            cbArxiv.Checked = false;
            rbDynamicColor.Checked = true;

            tfB0.Text = "0";
            tfB1.Text = "1";
            tfG0.Text = "1";
            tfG1.Text = "0";
            tfR0.Text = "0";
            tfR1.Text = "0";

        }
```

```
        #region Выбор секретного текста

        //показать выбор документа, содержащего скрываемый текст, по нажатию
на кнопку
        private void bGetFileSecretText_Click(object sender, EventArgs e)
        {
            showDilogOpenSecretText();
        }

        /// <summary>
        /// выбор документа с секретным текстом
        /// </summary>
        private void showDilogOpenSecretText()
        {
            OpenFileDialog openFileDialog = new OpenFileDialog();
            openFileDialog.Filter = "Документ (.doc, .docx, .rtf, .odt,
.txt)|*.docx;*.doc;*.rtf;*.odt;*.txt|Word документы|*.docx;*.doc;|Текстовые
файлы|*.txt|Все файлы|*.*";
            if (openFileDialog.ShowDialog() == DialogResult.OK)
            {
                openSecretText(openFileDialog.FileName);
            }
        }


        /// <summary>
        /// запись секретного текста из выбранного файла
        /// </summary>
        /// <param name="path">путь к файлу</param>
        private void openSecretText(String path)
        {

            Word.Document doc = openDoc(path);

            Object begin = Type.Missing;
            Object end = Type.Missing;

            Word.Range wordrange = doc.Range(ref begin, ref end);

            secretText = wordrange.Text;

            closeDoc(doc);
        }



        #endregion

        #region Изменение кодировки

        Encoding _currentEncoding;
        List<RadioButton> listRB;

        /// <summary>
        /// настройка радиобатонов отвечающих за кодировку
        /// </summary>
        private void setRBEncoding()
        {
            rEncoding32.Tag = Encoding.Unicode;
            rEncoding16.Tag = Encoding.UTF8;
            rEncoding8.Tag = Encoding.GetEncoding(1251);
            rEncoding7.Tag = Encoding.ASCII;
```

```
            listRB = new List<RadioButton>();
            listRB.Add(rEncoding32);
            listRB.Add(rEncoding16);
            listRB.Add(rEncoding8);
            listRB.Add(rEncoding7);

            foreach (RadioButton rb in listRB)
            {
                rb.CheckedChanged += changeRBEncoding;
            }

        }

        /// <summary>
        /// событие при выборе кодировки через радиобатон
        /// </summary>
        /// <param name="sender">объект инициализирующий событие</param>
        /// <param name="e">аргументы для события</param>
        private void changeRBEncoding(object sender, EventArgs e)
        {
            RadioButton rb = sender as RadioButton;
            if (rb.Checked && currentEncoding!=rb.Tag as Encoding)
            {
                currentEncoding = rb.Tag as Encoding;
            }
        }

        //кнопка "Автобвыбор" кодировки
        private void bAutoDetectEncoding_Click(object sender, EventArgs e)
        {
            autoDetectEncoding();
        }

        /// <summary>
        /// установка оптимальной кодировки
        /// </summary>
        private void autoDetectEncoding()
        {
            RadioButton rb = new RadioButton();

            Encoding unicode = Encoding.Unicode;

            String plainText = secretText;

            foreach(RadioButton r in listRB){
                String newText = getEncodingText(plainText, r.Tag as
Encoding);
                if (newText == plainText) rb = r;
            }

            currentEncoding = rb.Tag as Encoding;
        }

        /// <summary>
        /// аксессор для текущей кодировки
        /// </summary>
        public Encoding currentEncoding
        {
            get { return _currentEncoding; }
            set
            {
                _currentEncoding = value;

                foreach (RadioButton rb in listRB)
```

```
                {
                    if ((rb.Tag as Encoding) == currentEncoding &&
!rb.Checked)
                    {
                        rb.Checked = true;

                        break;
                    }
                }

                refreshCurrentText();


            }
        }


        #endregion

        #region Изменение секретного текста


        //прослушивание изменений в текстовом поле для секретного текста
        private void tfSecretText_TextChanged(object sender, EventArgs e)
        {
            secretTextChanged();
        }

        /// <summary>
        /// отловить изменения в скрываемом тексте
        /// </summary>
        private void secretTextChanged()
        {
            //запомнить, где был курсор во время изменений
            int ind = tfSecretText.SelectionStart;

            //обновить все связанные с текстом датчики
            refreshCurrentText();

            //вернуть указатель на исходную позицию
            tfSecretText.Select(ind, 0);

        }


        /// <summary>
        /// обновить все показатели, связанные с секретным текстом
        /// </summary>
        private void refreshCurrentText()
        {
            //обновить текст с учетом выбранной кодировки
            currentSecretText = getEncodingText(secretText, currentEncoding);

            labelCountChars.Text = currentSecretText.Length.ToString();

            //обновить двоичный вид текста
            currentBinSecretText = getBinEncodingText(currentSecretText,
currentEncoding);

            labelCountBit.Text = currentBinSecretText.Length.ToString();


            //отметить все символы, которые будут потеряны при текущей
кодировке
            selectFailChar(secretText, currentSecretText);
```

```
        }

        /// <summary>
        /// получить текст в текущей кодировке
        /// </summary>
        /// <param name="plainText">исходный текст</param>
        /// <param name="currentEncoding">необходимая кодировка</param>
        /// <returns>текст после преобразования</returns>
        private string getEncodingText(String plainText, Encoding encoding)
        {
            //кодировка оригинального текста
            Encoding unicode = Encoding.Unicode;

            byte[] unicodeBytes = unicode.GetBytes(secretText);
            byte[] currentBytes = Encoding.Convert(unicode, encoding,
unicodeBytes);


            char[] currentEncodingChars = new
char[encoding.GetCharCount(currentBytes, 0, currentBytes.Length)];
            encoding.GetChars(currentBytes, 0, currentBytes.Length,
currentEncodingChars, 0);
            string e1251String = new string(currentEncodingChars);

            return e1251String;

        }

        /// <summary>
        /// получить битовое представление текста
        /// </summary>
        /// <param name="plainText">исходный текст</param>
        /// <param name="encoding">кодировка</param>
        /// <returns>строка с битовым представлением</returns>
        private string getBinEncodingText(String plainText, Encoding
encoding)
        {
            byte[] currentBytes = encoding.GetBytes(plainText);
            String strBin = "";
            String space = (encoding == Encoding.ASCII) ? "0000000" :
"00000000";

            for (var i = 0; i < currentBytes.Length; i++)
            {
                string bin = Convert.ToString(currentBytes[i], 2);

                if (bin.Length != space.Length)
                {
                    if (bin.Length < space.Length)
                    {
                        bin = space.Substring(bin.Length) + bin;
                    }
                    else
                    {
                        bin = bin.Substring(0, space.Length);
                    }

                }

                strBin += bin;
            }
            return strBin;
        }

        /// <summary>
```

```
        /// сравнить символы в указанных строках, и отметить несовпадающие
        /// </summary>
        /// <param name="before"></param>
        /// <param name="after"></param>
        private void selectFailChar(String before, String after)
        {
            tfCurrentSecretText.SelectAll();
            tfCurrentSecretText.SelectionBackColor =
Color.FromKnownColor(KnownColor.Control);

            tfSecretText.SelectAll();
            tfSecretText.SelectionBackColor = Color.White;

            tfSecretText.Select(0, 0);

            //успешно ли переведен текст
            Boolean isGood = true;

            for (int i = 0; i < before.Length; i++)
            {
                if (before[i] != after[i])
                {
                    tfCurrentSecretText.Select(i, 1);
                    tfCurrentSecretText.SelectionBackColor =
System.Drawing.Color.Aqua;

                    tfSecretText.Select(i, 1);
                    tfSecretText.SelectionBackColor =
System.Drawing.Color.Aqua;

                    isGood = false;
                }
            }

            if (!isGood)
            {
                labelWarring.Text = "некоторые символы будут потеряны";
            }
            else
            {
                labelWarring.Text = "";
            }
        }



        #endregion

        #region Аксессоры для текстовых полей

        //следит за тем, изменялся ли текст или нет (нужно для того, чтобы не
делать лишний раз архивацию)
        Boolean isChangeText = false;

        /// <summary>
        /// работа с текстом для для скрытия
        /// </summary>
        public String secretText
        {
            get { return tfSecretText.Text; }
            set { tfSecretText.Text = value; isChangeText = true; }
        }

        public String currentSecretText
        {
```

```
        get { return tfCurrentSecretText.Text; }
        set { tfCurrentSecretText.Text = value; }
    }

    public String currentBinSecretText
    {
        get { return tfCurrentBinSecretText.Text; }
        set { tfCurrentBinSecretText.Text = value; }
    }

    #endregion

    #region Заставка

    Preloader preloader;

    /// <summary>
    /// загрузка необходимых библиотек, MS Office Word с заставкой
    /// </summary>
    private void preset()
    {
        showLogo();
        initOfficeWord();
        hideLogo();
    }

    private void showLogo()
    {
        this.Visible = false;
        preloader = new Preloader();
        preloader.Show();
    }
    private void hideLogo()
    {
        preloader.Close();
        this.Visible = true;
        this.Activate();
    }


    #endregion

    #region Архивация

    //ключ для внешнего режима архивации
    String strKeyArxiv;

    //эффект сжатия
    int profit;

    //текст после сжатия
    String binArxiv;


    //смена работы архивации через чекбокс
    private void cbArxiv_CheckedChanged(object sender, EventArgs e)
    {
        changeArxivView(cbArxiv.Checked);
    }

    //изменение режима архивации
    private void changeArxivView(Boolean view)
    {
        if (view != cbArxiv.Checked) cbArxiv.Checked = view;
        else
```

```
    {
        if (view)
        {
            panelArxiv.Enabled = true;

            getArxivText();

        }
        else
        {
            panelArxiv.Enabled = false;
        }
    }

}

//вызов архивации текста "Обновить"
private void button8_Click(object sender, EventArgs e)
{
    getArxivText();

}

//при смене режима архивации через радибатоны
private void rArxivRbChange(object sender, EventArgs e)
{
    if ((sender as RadioButton).Checked)
    {
        showArxivEffect();
    }
}

// показать результаты архивации
private void showArxivEffect()
{
    if (rArxiveKey.Checked)
    {
        tfKeyArxiv.Text = strKeyArxiv;
    }
    else
    {
        tfKeyArxiv.Text = "";
    }

    int rp = (rArxiveKey.Checked) ? profit : profit - 16;
    labelProfit.Text = rp.ToString();

}

//архивация текста
private void getArxivText()
{
    String arx = BY.getBY(tfCurrentBinSecretText.Text);
    int indexSep = arx.LastIndexOf('|');


    strKeyArxiv = arx.Substring(indexSep + 1);

    binArxiv = arx.Substring(0, indexSep);

    profit = countProfit(binArxiv);

    Console.WriteLine("end = "+replaceArxiv(binArxiv)+" ---");
    isChangeText = false;
```

```
    showArxivEffect();
}

/// <summary>
/// общий подсчет выгоды от архивации
/// </summary>
/// <param name="str">исходная строка</param>
/// <returns>результат</returns>
private int countProfit(String str)
{
    Console.WriteLine("--- count profit ---");
    Console.WriteLine("beg = "+str+" ---");
    int profit = 0;
    if (str.Length > 0)
    {
        char c = str[0];
        int equal = 1;
        for (int i = 1; i < str.Length; i++)
        {
            if (str[i] == c) equal++;
            else
            {
                profit += findRealPrior(equal);
                equal = 1;
                c = str[i];
            }
        }

        profit += findRealPrior(equal);
    }

    profit = currentBinSecretText.Length - profit;

    return profit;
}


/// <summary>
/// подсчет выгоды от сжатие последовательности заданной длинны
/// </summary>
/// <param name="equal">количество символов подряд</param>
/// <returns></returns>
private int findRealPrior(int equal)
{
    int prior = 0;
    int len = 0;
    int n = equal;
    int[] ps = new int[4] { 6, 5, 4, 3 };

    foreach (int p in ps)
    {
        int count = n / p;
        n -= count * p;
        len += count * 2;
    }
    len += n;

    return len;
}


#endregion

#region Настройка цвета
```

```
        //событие при установке какого либо значения радиобатонов цвета
        private void rColorChanged(object sender, EventArgs e)
        {
            RadioButton rb = sender as RadioButton;
            if (rb.Checked)
            {
                setEnableGroupColor(rbDynamicColor.Checked);

            }
        }


        private void setEnableGroupColor(Boolean viewDynamicColor)
        {
            groupBoxDynamicColor.Enabled = viewDynamicColor;
            groupBoxStaticColor.Enabled = !viewDynamicColor;
        }


        #region Настройка статического цвета
        //установка статического цвета для 1
        private void bColorSetOne_Click(object sender, EventArgs e)
        {
            setColorField(tfSR1, tfSG1, tfSB1);
        }

        //установка статического цвета для 0
        private void bColorSetZero_Click(object sender, EventArgs e)
        {
            setColorField(tfSR0, tfSG0, tfSB0);
        }

        /// <summary>
        /// настройка текстовых полей цвета с помощью вызова ColorPicker
        /// </summary>
        /// <param name="tfR">красный</param>
        /// <param name="tfG">зеленый</param>
        /// <param name="tfB">синий</param>
        private void setColorField(TextBox tfR, TextBox tfG, TextBox tfB)
        {
            if (tfR.Text == "") tfR.Text = "0";
            if (tfG.Text == "") tfG.Text = "0";
            if (tfB.Text == "") tfB.Text = "0";

            byte red = Convert.ToByte(tfR.Text);
            byte green = Convert.ToByte(tfG.Text);
            byte blue = Convert.ToByte(tfB.Text);


            Color currentColor = Color.FromArgb(red, green, blue);
Color cd = showColorPicker(currentColor);

            tfB.Text = cd.B.ToString();
            tfR.Text = cd.R.ToString();
            tfG.Text = cd.G.ToString();
        }

        /// <summary>
        /// показать ColorPicker с заданным цветом, и вернуть выбранный
        /// </summary>
        /// <param name="color"></param>
        /// <returns></returns>
        private Color showColorPicker(Color color)
        {
            ColorDialog cd = new ColorDialog();
```

```
            cd.Color = color;
            cd.ShowDialog();
            return cd.Color;
        }

        #endregion

        #endregion

        #region Работа с Word

        Word.Application officeWord;//программа Office Word
        Word.Document doc;//документ

        String pathDoc;

        #region Функции открытия / закрытия

        /// <summary>
        /// загрузка программы Microsoft Office Word
        /// </summary>
        private void initOfficeWord()
        {
            officeWord = new Microsoft.Office.Interop.Word.Application();
            officeWord.DocumentBeforeClose += new
Microsoft.Office.Interop.Word.ApplicationEvents4_DocumentBeforeCloseEventHand
ler(wa_DocumentBeforeClose);
        }

        /// <summary>
        /// закрытие приложения Microsoft Office Word
        /// </summary>
        private void exitOfficeWord()
        {
            officeWord.DocumentBeforeClose -= new
Microsoft.Office.Interop.Word.ApplicationEvents4_DocumentBeforeCloseEventHand
ler(wa_DocumentBeforeClose);

            if (doc != null)
            {
                closeDoc(doc);
            }

            object sch = Type.Missing;
            object aq = Type.Missing;
            object ab = Type.Missing;
            (officeWord as
Microsoft.Office.Interop.Word._Application).Quit(ref sch, ref aq, ref ab);
        }

        /// <summary>
        /// открыть документ Word
        /// </summary>
        /// <param name="path">путь к документу</param>
        /// <returns></returns>
        private Microsoft.Office.Interop.Word.Document openDoc(string path)
        {
            Object filename = path;
            Object confirmConversions = Type.Missing;
            Object readOnly = Type.Missing;
            Object addToRecentFiles = Type.Missing;
            Object passwordDocument = Type.Missing;
            Object passwordTemplate = Type.Missing;
            Object revert = Type.Missing;
            Object writePasswordDocument = Type.Missing;
```

```
            Object writePasswordTemplate = Type.Missing;
            Object format = Type.Missing;
            Object encoding = Type.Missing;
            Object visible = Type.Missing;
            Object openConflictDocument = Type.Missing;
            Object openAndRepair = Type.Missing;
            Object documentDirection = Type.Missing;
            Object noEncodingDialog = Type.Missing;


            officeWord.Documents.Open(ref filename,
            ref confirmConversions,
            ref readOnly,
            ref addToRecentFiles,
            ref passwordDocument,
            ref passwordTemplate,
            ref revert,
            ref writePasswordDocument,
            ref writePasswordTemplate,
            ref format,
            ref encoding,
            ref visible,
            ref openConflictDocument,
            ref openAndRepair,
            ref documentDirection,
            ref noEncodingDialog);

            Word.Document doc =
officeWord.Documents.Application.ActiveDocument;

            return doc;
        }

        /// <summary>
        /// закрытие документа Microsoft Office Word
        /// </summary>
        /// <param name="doc">объект хранящий ссылку на документ</param>
        private void closeDoc(Microsoft.Office.Interop.Word.Document doc)
        {

            object sch = Word.WdSaveOptions.wdDoNotSaveChanges;
            object aq = Type.Missing;
            object ab = Type.Missing;

            (doc as Microsoft.Office.Interop.Word._Document).Close(ref sch,
ref aq, ref ab);
        }

        #endregion

        #region Прослушка собственного закрытия Word-а

        delegate void SetTextCallback(bool flag);
        private Thread demoThread = null;

        ////////////////////////////////////
        private void wa_DocumentBeforeClose(Word.Document Doc, ref bool
Cancel)
        {
            closeDoc(doc);
            officeWord.Visible = false;
            this.demoThread = new Thread(new
ThreadStart(this.ThreadProcSafe));
            this.demoThread.Start();
        }
```

```
        private void ThreadProcSafe()
        {
            this.changeAfterUserClose(false);
        }

        private void changeAfterUserClose(bool flag)
        {
            if (this.pictureBoxWordActive.InvokeRequired)
            {
                SetTextCallback d = new
SetTextCallback(changeAfterUserClose);
                this.Invoke(d, new object[] { flag });
            }
            else
            {
                changeViewWordFace(flag);
                doc = null;
            }
        }

        #endregion

        #region Управление Word

        private void changeViewWordFace(bool view)
        {
            pictureBoxWordDesactive.Visible = !view;
            panelOper.Visible = view;
            panelWord.Visible = view;

            bImplant.Visible = view;
            bExtract.Visible = view;

            bCloseWord.Visible = view;

        }

        //показать водр
        private void bShowDoc_Click(object sender, EventArgs e)
        {
            showDoc();
        }

        private void showDoc()
        {
            if (officeWord.Visible)
            {
                officeWord.Visible = false;
                bShowDoc.Text = "показать";

            }
            else
            {
                officeWord.Visible = true;
                bShowDoc.Text = "скрыть";

            }
        }

        private void loadWord(string path)
        {
            Console.WriteLine(path);
            doc = openDoc(path);
            changeViewWordFace(true);
```

```
        }

        //выбор файла для защиты
        private void bGetDoc_Click(object sender, EventArgs e)
        {

            OpenFileDialog openFileDialog = new OpenFileDialog();
            openFileDialog.Filter = "Документы Word|*.docx;*.doc;|rtf,
odt|*.rtf;*.odt|Все файлы|*.*";
            if (openFileDialog.ShowDialog() == DialogResult.OK)
            {

                if (doc != null)
                {
                    // MessageBox.Show("close doc");
                    closeDoc(doc);
                }
                pathDoc = openFileDialog.FileName;
                loadWord(pathDoc);

                labelWordName.Text =
System.IO.Path.GetFileNameWithoutExtension(pathDoc);

                object objMissing = System.Reflection.Missing.Value;
                labelWordChars.Text =
doc.ComputeStatistics(Word.WdStatistic.wdStatisticCharactersWithSpaces, ref
objMissing).ToString();

                labelWordCharsWithoutSpaces.Text =
doc.ComputeStatistics(Word.WdStatistic.wdStatisticCharacters, ref
objMissing).ToString();

                changeViewWordFace(true);

            }
        }


        private void bSaveAsDoc_Click(object sender, EventArgs e)
        {

            SaveFileDialog saveFileDialog = new SaveFileDialog();
            saveFileDialog.FileName = pathDoc;
            if (saveFileDialog.ShowDialog() == DialogResult.OK)
            {
                saveDoc(saveFileDialog.FileName);
            }

        }


        private void bSaveDoc_Click(object sender, EventArgs e)
        {
            saveDoc(pathDoc);
        }

        private void saveDoc(string path)
        {
            Object fileName = path;
            Object fileFormat = Type.Missing;
            Object lockComments = Type.Missing;
            Object password = Type.Missing;
            Object addToRecentFiles = Type.Missing;
            Object writePassword = Type.Missing;
            Object readOnlyRecommended = Type.Missing;
```

```
            Object embedTrueTypeFonts = Type.Missing;
            Object saveNativePictureFormat = Type.Missing;
            Object saveFormsData = Type.Missing;
            Object saveAsAOCELetter = Type.Missing;
            Object encoding = Type.Missing;
            Object insertLineBreaks = Type.Missing;
            Object allowSubstitutions = Type.Missing;
            Object lineEnding = Type.Missing;
            Object addBiDiMarks = Type.Missing;

            doc.SaveAs(ref fileName, ref fileFormat, ref lockComments,
            ref password, ref addToRecentFiles, ref writePassword,
            ref readOnlyRecommended, ref embedTrueTypeFonts,
            ref saveNativePictureFormat, ref saveFormsData,
            ref saveAsAOCELetter, ref encoding, ref insertLineBreaks,
            ref allowSubstitutions, ref lineEnding, ref addBiDiMarks);
        }


        //закрытие документа Word с помощью кнопки "x"
        private void bCloseWord_Click(object sender, EventArgs e)
        {
            changeViewWordFace(false);
        }

        #endregion

        #endregion

        #region Внедрение текста

        //набор индексов выбранных для скрытия текста
        List<int> indexs;

        //обработка нажатия на кнопку "Внедрить
        private void bImplant_Click(object sender, EventArgs e)
        {
            startImplantText();
        }

        /// <summary>
        /// получить текст для внедрения
        /// </summary>
        /// <returns></returns>
        private String getImplantText()
        {
            String text = "";

            if (cbArxiv.Checked)
            {
                //если текст был изменен, но обновления информации для
архивации не было
                if(isChangeText) getArxivText();

                if (rArxivInside.Checked)
                {
                    String strNum =
Convert.ToString(Convert.ToInt16(strKeyArxiv), 2);
                    Console.WriteLine("strNum = " + strNum);
                    if (strNum.Length != 16)
                    {
                        string space = "0000000000000000";
                        strNum = space.Substring(strNum.Length) + strNum;
                    }
                    Console.WriteLine("strNum add = " + strNum);
```

```
                text = strNum + binArxiv;
            }
            else
            {
                text = replaceArxiv(binArxiv);
                Console.WriteLine("strResult Implant text = " + text);
            }
        }
        else
        {
            text = currentBinSecretText;
        }

        return text;
    }

    /// <summary>
    /// обработка архива
    /// </summary>
    /// <param name="binArxiv">бинарный вид</param>
    /// <returns>результат</returns>
    private string replaceArxiv(string binArxiv)
    {
        String result = "";
        int count = 1;
        char c=binArxiv[0];
        int len = binArxiv.Length;
        for (int i = 1; i < len; i++)
        {
            if (binArxiv[i] == c)
            {
                 count++;
                if(count==6 || i==len-1){
                    result+=getArxivCharsReplace(count,c);
                    count=0;
                }
            }
            else
            {
                result += getArxivCharsReplace(count, c);
                count = 1;
                c = binArxiv[i];
                if (i == len - 1) result += getArxivCharsReplace(count,
c);
            }
        }
        if (count != 1) result += getArxivCharsReplace(count, c);




        return result;
    }

    /// <summary>
    /// обработка данных для рахивации
    /// </summary>
    /// <param name="count">количество идущих подряд символов</param>
    /// <param name="c">символ</param>
    /// <returns></returns>
    private string getArxivCharsReplace(int count, char c)
    {
        String result = "";
        string cc = (c == '0') ? "A" : "B";
        switch (count)
```

```
            {
                case 6: result += cc + "B"; break;
                case 5: result += cc + "A"; break;
                case 4: result += cc + "1"; break;
                case 3: result += cc + "0"; break;
                default: for (int j = 0; j < count; j++) result +=
c.ToString(); break;
            }
            return result;
        }


        /// <summary>
        /// внедрение символов в  документ
        /// </summary>
        private void startImplantText()
        {
            //проверить что есть текст для скрытия
            if (secretText.Length > 0)
            {
                if (checkColor())
                {
                    //текст для внедрения
                    String implantText = getImplantText();

                    //минимальное расстояние между символами
                    int minStep = (rbDynamicColor.Checked) ? 2 : 1;

                    object objMissing = System.Reflection.Missing.Value;

                    int countCharsDoc = (rPrint.Checked) ?
doc.ComputeStatistics(Microsoft.Office.Interop.Word.WdStatistic.wdStatisticCh
aracters, ref objMissing) :
doc.ComputeStatistics(Microsoft.Office.Interop.Word.WdStatistic.wdStatisticCh
aractersWithSpaces, ref objMissing);
                    int countCharsBin = implantText.Length;

                    int maxStep = countCharsDoc / (countCharsBin * minStep);

                    if (maxStep > 1)
                    {
                        //получить список индексов для внедрения
                        indexs = getListIndex(implantText, countCharsDoc,
countCharsBin, minStep, maxStep);

                        if (indexs != null)
                        {

                            applyImplant(implantText);

                        }
                        else
                        {
                            MessageBox.Show("Не удалось внедрить текст");
                        }
                    }
                    else
                    {
                        MessageBox.Show("В файле не достаточно символов для
скрытия\nдобавьте еще " + (countCharsBin * minStep - countCharsDoc) + "
символ");
                    }
                }
                else
                {
```

```
                    MessageBox.Show("Установите правильные значения для
цвета!");
                }
            }
            else
            {
                MessageBox.Show("Введите текст для скрытия!");
            }
        }


        /// <summary>
        /// проверка цвета при извлечении данных
        /// </summary>
        /// <returns></returns>
        private bool checkColor()
        {
            bool flagGood = true;
            if (rbDynamicColor.Checked)
            {
                if (tfR0.Text == tfR1.Text && tfG0.Text == tfG1.Text &&
tfB0.Text == tfB1.Text)
                {
                    MessageBox.Show("Внимание. Значения отклонений цвета для
символов 0 и 1 должны отличаться хотя бы в одним полем");
                    flagGood = false;
                }
                if (tfR0.Text == "" | tfG0.Text == "" | tfB0.Text == "" |
tfR1.Text == "" | tfG1.Text == "" | tfB1.Text == "")
                {
                    MessageBox.Show("Внимание. Все поля должны быть
заполнены");
                    flagGood = false;
                }
            }
            else
            {
                if (tfSR0.Text == tfSR1.Text && tfSG0.Text == tfSG1.Text &&
tfSB0.Text == tfSB1.Text)
                {
                    MessageBox.Show("Внимание. Значения отклонений цвета для
символов 0 и 1 должны отличаться хотя бы в одним полем");
                    flagGood = false;
                }
                if (tfSR0.Text == "" | tfSG0.Text == "" | tfSB0.Text == "" |
tfSR1.Text == "" | tfSG1.Text == "" | tfSB1.Text == "")
                {
                    MessageBox.Show("Внимание. Все поля должны быть
заполнены");
                    flagGood = false;
                }
            }

            if (cbArxiv.Checked)
            {
                if (tfAR0.Text == tfAR1.Text && tfAG0.Text == tfAG1.Text &&
tfAB0.Text == tfAB1.Text)
                {
                    MessageBox.Show("Внимание. Значения дополнительного цвета
для архива должны быть отличны хотя бы в одном поле");
                    flagGood = false;
                }
                if (tfAR0.Text == "" | tfAG0.Text == "" | tfAB0.Text == "" |
tfAR1.Text == "" | tfAG1.Text == "" | tfAB1.Text == "")
                {
```

```
                    MessageBox.Show("Внимание. Все поля с архивным цветом
должны быть заполнены");
                    flagGood = false;
                }

                List<TextBox> tfA = new List<TextBox>();

tfA.Add(tfAR0);tfA.Add(tfAG0);tfA.Add(tfAB0);tfA.Add(tfAR1);tfA.Add(tfAG1);tf
A.Add(tfAB1);
                List<TextBox>tfD = new List<TextBox>();

tfD.Add(tfR0);tfD.Add(tfG0);tfD.Add(tfB0);tfD.Add(tfR1);tfD.Add(tfG1);tfD.Add
(tfB1);
                List<TextBox>tfS = new List<TextBox>();

tfS.Add(tfSR0);tfS.Add(tfSG0);tfS.Add(tfSB0);tfS.Add(tfSR1);tfS.Add(tfSG1);tf
S.Add(tfSB1);


                List<TextBox> tt = (rbDynamicColor.Checked)?tfD:tfS;

                if (tfA[0].Text == tt[0].Text && tfA[1].Text == tt[1].Text &&
tfA[2].Text == tt[2].Text)
                {
                    MessageBox.Show("Внимание. Значение архивного цвета не
должно сопадать с основным");
                    flagGood = false;
                }

                if (tfA[3].Text == tt[3].Text && tfA[4].Text == tt[4].Text &&
tfA[5].Text == tt[5].Text)
                {
                    MessageBox.Show("Внимание. Значение архивного цвета не
должно сопадать с основным");
                    flagGood = false;
                }



            }

            return flagGood;
        }

        /// <summary>
        /// внедрение секретного текста в документ
        /// </summary>
        /// <param name="strBin">двоичный вид секретного текста</param>
        private void applyImplant(String strBin)
        {
            //индикаторы текущего символа
            Object startA, endA, startB, endB;

            Color colorA, colorB;

            //составляющие цвета
            int red, green, blue;
            red = green = blue = 0;

            int red0, green0, blue0, red1, green1, blue1;
            red0 = green0 = blue0 = red1 = green1 = blue1 = 0;

            byte redA0, greenA0, blueA0, redA1, greenA1, blueA1;
            redA0 = greenA0 = blueA0 = redA1 = greenA1 = blueA1 = 0;
```

```
            if (cbArxiv.Checked)
            {
                redA0 = Convert.ToByte(tfAR0.Text);
                greenA0 = Convert.ToByte(tfAG0.Text);
                blueA0 = Convert.ToByte(tfAB0.Text);

                redA1 = Convert.ToByte(tfAR1.Text);
                greenA1 = Convert.ToByte(tfAG1.Text);
                blueA1 = Convert.ToByte(tfAB1.Text);
            }

            if (rbStaticColor.Checked)
            {
                red0 = Convert.ToByte(tfSR0.Text);
                green0 = Convert.ToByte(tfSG0.Text);
                blue0 = Convert.ToByte(tfSB0.Text);

                red1 = Convert.ToByte(tfSR1.Text);
                green1 = Convert.ToByte(tfSG1.Text);
                blue1 = Convert.ToByte(tfSB1.Text);
            }

            byte shiftR1 = Convert.ToByte(tfR1.Text);
            byte shiftG1 = Convert.ToByte(tfG1.Text);
            byte shiftB1 = Convert.ToByte(tfB1.Text);
            byte shiftR0 = Convert.ToByte(tfR0.Text);
            byte shiftG0 = Convert.ToByte(tfG0.Text);
            byte shiftB0 = Convert.ToByte(tfB0.Text);



            for (var i = 0; i < strBin.Length; i++)
            {

                if (i % 8 == 0)
                {
                    labelProgress.Text = i + "/" + strBin.Length;
                    labelProgress.Refresh();
                }

                int pos = indexs[i];


                startB = (int)pos;
                endB = (int)startB + 1;

                Word.Range rngB = doc.Range(ref startB, ref endB);
                colorB = ToColor(rngB.Font.Color);

                //внедрить динамический цвет
                if (rbDynamicColor.Checked)
                {
                    startA = pos - 1;
                    endA = (int)startA + 1;

                    Word.Range rngA = doc.Range(ref startA, ref endA);
                    colorA = ToColor(rngA.Font.Color);



                    if (strBin[i] == '0')
                    {
                        red = (colorA.R + shiftR0 > 255) ? colorA.R - shiftR0
: colorA.R + shiftR0;
```

```
                        green = (colorA.G + shiftG0 > 255) ? colorA.G -
shiftG0 : colorA.G + shiftG0;
                        blue = (colorA.B + shiftB0 > 255) ? colorA.B -
shiftB0 : colorA.B + shiftB0;

                    }
                    else if (strBin[i] == '1')
                    {
                        red = (colorA.R + shiftR1 > 255) ? colorA.R - shiftR1
: colorA.R + shiftR1;
                        green = (colorA.G + shiftG1 > 255) ? colorA.G -
shiftG1 : colorA.G + shiftG1;
                        blue = (colorA.B + shiftB1 > 255) ? colorA.B -
shiftB1 : colorA.B + shiftB1;
                    }
                    else if (strBin[i] == 'A')
                    {
                        red = (colorA.R + redA0 > 255) ? colorA.R - redA0 :
colorA.R + redA0;
                        green = (colorA.G + greenA0 > 255) ? colorA.G -
greenA0 : colorA.G + greenA0;
                        blue = (colorA.B + blueA0 > 255) ? colorA.B - blueA0
: colorA.B + blueA0;
                    }
                    else if (strBin[i] == 'B')
                    {
                        red = (colorA.R + redA1 > 255) ? colorA.R - redA1 :
colorA.R + redA1;
                        green = (colorA.G + greenA1 > 255) ? colorA.G -
greenA1 : colorA.G + greenA1;
                        blue = (colorA.B + blueA1 > 255) ? colorA.B - blueA1
: colorA.B + blueA1;
                    }

                    rngB.Font.Color =
(Microsoft.Office.Interop.Word.WdColor)Microsoft.VisualBasic.Information.RGB(
red, green, blue);
                }
                else
                {
                    if (strBin[i] == '0')
                    {
                        rngB.Font.Color =
(Microsoft.Office.Interop.Word.WdColor)Microsoft.VisualBasic.Information.RGB(
red0, green0, blue0);
                    }
                    else if (strBin[i] == '1')
                    {
                        rngB.Font.Color =
(Microsoft.Office.Interop.Word.WdColor)Microsoft.VisualBasic.Information.RGB(
red1, green1, blue1);
                    }
                    else if (strBin[i] == 'A')
                    {
                        rngB.Font.Color =
(Microsoft.Office.Interop.Word.WdColor)Microsoft.VisualBasic.Information.RGB(
redA0, greenA0, blueA0);
                    }
                    else if (strBin[i] == 'B')
                    {
                        rngB.Font.Color =
(Microsoft.Office.Interop.Word.WdColor)Microsoft.VisualBasic.Information.RGB(
redA1, greenA1, blueA1);
                    }
                }
```

```
            }

            labelProgress.Text = "";

            MessageBox.Show("текст успешно внедрен");
        }


        /// <summary>
        /// создать список симоволов, в которые будет производиться скрытие
        /// </summary>
        /// <param name="strBin">строка для скрытия</param>
        /// <param name="countCharsDoc">число символов в документе</param>
        /// <param name="countCharsBin">число символов в скрытом
тексте</param>
        /// <param name="minStep">минимальный шаг разброса</param>
        /// <param name="maxStep">максимальный шаг разброса</param>
        /// <returns>набор результирующих индексов</returns>
        private List<int> getListIndex(string strBin, int countCharsDoc, int
countCharsBin, int minStep, int maxStep)
        {
            int COUNT_ATTEMP = 100;
            List<int> indexs = new List<int>();

            Object startA, endA, startB, endB;

            startA = -2;

            Random wRnd = new Random(DateTime.Now.Millisecond);

            //внедрение символов
            int countAttempt = 0;

            Boolean isLimit = false;

            Console.WriteLine("maxStep = " + maxStep);


            do
            {

                startA = (int)0;
                endA = (int)0;
                startB = (int)0;
                endB = (int)0;

                isLimit = false;
                indexs = new List<int>();
                int shift = maxStep * 2 - minStep - countAttempt;
                if (shift < minStep) shift = minStep;
                countAttempt++;

                Console.WriteLine("attemp " + countAttempt);

                for (int i = 0; i < strBin.Length && !isLimit; i++)
                {
                    //расстояние для следующего символа
                    int step = wRnd.Next(minStep, shift);

                    Console.Write(step + " ");

                    startA = (int)startA + step;
                    endA = (int)startA + 1;
                    startB = (int)endA;
                    endB = (int)startB + 1;
```

```
                    if ((int)startB < countCharsDoc)
                    {
                        //предыдущий символ
                        Word.Range rngA = doc.Range(ref startA, ref endA);

                        //изменяемый символ
                        Word.Range rngB = doc.Range(ref startB, ref endB);

                        //необходимо чтобы предыдущий символ не был
пробельным, а также если выбран режим печати, чтобы пробельным не был и
изменяемый
                        while (!isLimit && checkChars(rngA.Text) &&
rPrint.Checked && checkChars(rngB.Text))
                        {
                            startA = (int)startA + 1;
                            endA = (int)startA + 1;
                            startB = (int)endA;
                            endB = (int)startB + 1;



                            if ((int)startB >= countCharsDoc)
                            {
                                isLimit = true;
                                //MessageBox.Show("LIMIT");

                            }
                            else
                            {
                                rngA = doc.Range(ref startA, ref endA);
                                rngB = doc.Range(ref startB, ref endB);
                            }
                        }

                        indexs.Add((int)startB);
                    }
                    else
                    {
                        //  MessageBox.Show("LIMIT");
                        isLimit = true;

                    }
                }
            } while (isLimit && countAttempt < COUNT_ATTEMP);

            if (countAttempt == COUNT_ATTEMP) indexs = null;

            return indexs;
        }

        /// <summary>
        /// проверка, не является ли символ невидимым (в случае печатного
режима)
        /// </summary>
        /// <param name="str">данные</param>
        /// <returns>результат</returns>
        private bool checkChars(string str)
        {
            Boolean flag = true;
            if (str == " " || str == "\t" || str == "\n" || str == "\r") flag
= false;
            return flag;
        }
```

```
        #endregion

        #region Обработка изменных символов

        Boolean isMarked = false;

        private void bViewImplant_Click(object sender, EventArgs e)
        {
            changeMarkedChars();
        }

        private void bDeleteImplantChars_Click(object sender, EventArgs e)
        {
            clearImplantChars();
        }


        /// <summary>
        /// маркировка символов содержащих скрытый текст
        /// </summary>
        private void changeMarkedChars()
        {
            if (indexs != null)
            {
                Object startA, endA, startB, endB;
                Word.Range rngA, rngB;
                Boolean clear = isMarked;
                foreach (int index in indexs)
                {
                    startA = index;
                    endA = (int)startA + 1;
                    rngA = doc.Range(ref startA, ref endA);
                    if (clear)
                    {
                        endB = startA;
                        startB = (int)endB - 1;
                        rngB = doc.Range(ref startB, ref endB);
                        rngA.Font.Shading.BackgroundPatternColorIndex =
rngB.Font.Shading.BackgroundPatternColorIndex;
                    }
                    else rngA.Font.Shading.BackgroundPatternColorIndex =
Word.WdColorIndex.wdBlue;
                }
                if (clear)
                {
                    bViewImplant.Text = "отметить";
                }
                else
                {
                    bViewImplant.Text = "очисить";
                }
                isMarked = !isMarked;
            }
        }

        /// <summary>
        /// внедрение данных
        /// </summary>
        private void clearImplantChars()
        {
            if (indexs != null)
            {
                Object startA, endA, startB, endB;
                Word.Range rngA, rngB;
```

```
                int i = 0;
                int count = indexs.Count;

                foreach (int index in indexs)
                {
                    if (i % 10 == 0)
                    {
                        labelProgress.Text = i + "/" + count;
                        labelProgress.Refresh();
                    }

                    startA = index;
                    endA = (int)startA + 1;
                    rngA = doc.Range(ref startA, ref endA);

                        endB = startA;
                        startB = (int)endB - 1;
                        rngB = doc.Range(ref startB, ref endB);
                        rngA.Font.Color = rngB.Font.Color;

                }
                labelProgress.Text = "";
            }
        }



        #endregion

        #region Извлечение текста

        private void bExtract_Click(object sender, EventArgs e)
        {
            startExtractText();
        }

        /// <summary>
        /// извлечение данных из документа
        /// </summary>
        private void startExtractText()
        {
            labelProgress.Text = "извлечение..";
            if (checkColor())
            {
                String binText = (rbDynamicColor.Checked) ?
extractBinDynamic() : extractBinStatic();
                labelProgress.Text = "";

                Console.WriteLine("bin = " + binText);

                int size = (currentEncoding == Encoding.ASCII) ? 7 : 8;
                if (cbArxiv.Checked)
                {
                    binText = unArxivText(binText);
                }
                String extractText = binToString(binText, currentEncoding,
size);


                    secretText = extractText;

                if (secretText.Length == 0) MessageBox.Show("при данных
настройка скрытого текста не обнаружено");
                else MessageBox.Show("текст извлечен");
```

```
            }
            else
            {
                MessageBox.Show("Укажите настройки цвета");
            }
        }

        /// <summary>
        /// разахривация данныех
        /// </summary>
        /// <param name="extractText">исходные данные</param>
        /// <returns>результат</returns>
        private string unArxivText(string extractText)
        {

            extractText = unCompress(extractText);
            int num = 0;
            if (rArxiveKey.Checked)
            {
                num = Convert.ToInt16(tfKeyArxiv.Text);
            }
            else
            {
                String strNum = extractText.Substring(0, 16);
                extractText = extractText.Substring(16);

                for (int i = 15; i > -1; i--)
                {
                    if (strNum[i] == '1') num +=
(int)(System.Math.Pow((double)2, (double)(15 - i)));
                }

            }



            return BY.extractBY(extractText, num);
        }


        /// <summary>
        /// декомпрессия данных
        /// </summary>
        /// <param name="extractText">исохдные данные</param>
        /// <returns>результат</returns>
        private string unCompress(string extractText)
        {
            string result = "";
            Console.WriteLine("extractText = "+extractText);
            for (int i = 0; i < extractText.Length; i++)
            {
                if (extractText[i] == '0' || extractText[i] == '1') result +=
extractText[i];
                else
                {
                    string c = (extractText[i]=='A')?"0":"1";
                    int count = 0;
                    switch (extractText[i + 1])
                    {
                        case '0': count = 3; break;
                        case '1': count = 4; break;
                        case 'A': count = 5; break;
                        case 'B': count = 6; break;
                    }
                    for (int j = 0; j < count; j++) result += c;
```

```
                    i++;
                }
            }
            Console.WriteLine("result = " + result);

           return result;
        }


        /// <summary>
        /// перевод двоичного вида в строковый
        /// </summary>
        /// <param name="strBin">исходные данные</param>
        /// <param name="encoding">кодировка</param>
        /// <param name="size">размер символа в кодировке</param>
        /// <returns></returns>
        private String binToString(String strBin, Encoding encoding, int
size)
        {
            byte[] arr = new byte[strBin.Length / size];
            for (int i = 0; i < strBin.Length; i += size)
            {
                string ss = strBin.Substring(i, size);
                byte bv = Convert.ToByte(ss, 2);
                arr[i / size] = bv;
            }
            string res = encoding.GetString(arr);

            return res;
        }

        /// <summary>
        ///  извлечение данных при статическом выборе цвета
        /// </summary>
        /// <returns></returns>
        private String extractBinStatic()
        {
            //индикаторы текущего символа
            indexs = new List<int>();
            int red0, green0, blue0, red1, green1, blue1;
            red0 = green0 = blue0 = red1 = green1 = blue1 = 0;

            byte redA0, greenA0, blueA0, redA1, greenA1, blueA1;
            redA0 = greenA0 = blueA0 = redA1 = greenA1 = blueA1 = 0;

            if (cbArxiv.Checked)
            {
                redA0 = Convert.ToByte(tfAR0.Text);
                greenA0 = Convert.ToByte(tfAG0.Text);
                blueA0 = Convert.ToByte(tfAB0.Text);

                redA1 = Convert.ToByte(tfAR1.Text);
                greenA1 = Convert.ToByte(tfAG1.Text);
                blueA1 = Convert.ToByte(tfAB1.Text);
            }

            red0 = Convert.ToByte(tfSR0.Text);
            green0 = Convert.ToByte(tfSG0.Text);
            blue0 = Convert.ToByte(tfSB0.Text);

            red1 = Convert.ToByte(tfSR1.Text);
            green1 = Convert.ToByte(tfSG1.Text);
            blue1 = Convert.ToByte(tfSB1.Text);

            Color color0 = Color.FromArgb(red0, green0, blue0);
```

```
            Color color1 = Color.FromArgb(red1, green1, blue1);
            Color colorAA = Color.FromArgb(redA0, greenA0, blueA0);
            Color colorBB = Color.FromArgb(redA1, greenA1, blueA1);


            String strBin = "";
            Object startA, endA;

            object objMissing = System.Reflection.Missing.Value;

            int count =
doc.ComputeStatistics(Microsoft.Office.Interop.Word.WdStatistic.wdStatisticCh
aractersWithSpaces, ref objMissing);


                for (var i = 0; i < count; i++)
                {
                    if (i % 50 == 0)
                    {
                        labelProgress.Text = i + "/" + count;
                        labelProgress.Refresh();
                    }

                    startA = i;
                    endA = (int)startA + 1;

                    Word.Range rngA = doc.Range(ref startA, ref endA);
                    Color colorA = ToColor(rngA.Font.Color);

                    if(colorA==color0){
                        strBin+="0";
                        indexs.Add(i);
                    }
                    else if(colorA==color1){
                        strBin+="1";
                        indexs.Add(i);
                    }
                    else if (colorA == colorAA)
                    {
                        strBin += "B";
                        indexs.Add(i);
                    }
                    else if (colorA == colorBB)
                    {
                        strBin += "B";
                        indexs.Add(i);
                    }


            }
                if (indexs.Count == 0) indexs = null;

            return strBin;
        }


        /// <summary>
        /// извлечение данных при динамическом подборе цвета
        /// </summary>
        /// <returns></returns>
        private String extractBinDynamic()
        {
            //индикаторы текущего символа
            indexs = new List<int>();
            String strBin = "";
```

```
            Object startA, endA, startB, endB;

            //составляющие цвета
            // int red, green, blue;

            Color colorA, colorB;

            byte redA0, greenA0, blueA0, redA1, greenA1, blueA1;
            redA0 = greenA0 = blueA0 = redA1 = greenA1 = blueA1 = 0;

            if (cbArxiv.Checked)
            {
                redA0 = Convert.ToByte(tfAR0.Text);
                greenA0 = Convert.ToByte(tfAG0.Text);
                blueA0 = Convert.ToByte(tfAB0.Text);

                redA1 = Convert.ToByte(tfAR1.Text);
                greenA1 = Convert.ToByte(tfAG1.Text);
                blueA1 = Convert.ToByte(tfAB1.Text);
            }

            byte shiftR1 = Convert.ToByte(tfR1.Text);
            byte shiftG1 = Convert.ToByte(tfG1.Text);
            byte shiftB1 = Convert.ToByte(tfB1.Text);
            byte shiftR0 = Convert.ToByte(tfR0.Text);
            byte shiftG0 = Convert.ToByte(tfG0.Text);
            byte shiftB0 = Convert.ToByte(tfB0.Text);


                byte shiftR, shiftG, shiftB;

                int count = doc.Characters.Count;


                for (var i = 0; i < count - 1; i++)
                {
                    if (i % 50 == 0)
                    {
                        labelProgress.Text = i + "/" + count;
                        labelProgress.Refresh();
                    }

                    startA = i;
                    endA = (int)startA + 1;
                    startB = (int)endA;
                    endB = (int)startB + 1;

                    Word.Range rngA = doc.Range(ref startA, ref endA);
                    colorA = ToColor(rngA.Font.Color);

                    Word.Range rngB = doc.Range(ref startB, ref endB);
                    colorB = ToColor(rngB.Font.Color);


                    if (colorA != colorB)
                    {

                        shiftR = (Convert.ToInt16(colorA.R) +
Convert.ToInt16(shiftR0) > 255) ? (byte)(colorA.R - colorB.R) :
(byte)(colorB.R - colorA.R);
                        shiftG = (Convert.ToInt16(colorA.G) +
Convert.ToInt16(shiftG0) > 255) ? (byte)(colorA.G - colorB.G) :
(byte)(colorB.G - colorA.G);
```

```
                        shiftB = (Convert.ToInt16(colorA.B) +
Convert.ToInt16(shiftB0) > 255) ? (byte)(colorA.B - colorB.B) :
(byte)(colorB.B - colorA.B);

                        if (shiftR == shiftR1 && shiftG1 == shiftG && shiftB
== shiftB1)
                        {
                            strBin += '1';
                            indexs.Add(i + 1);
                        }
                        else if (shiftR == shiftR0 && shiftG0 == shiftG &&
shiftB == shiftB0)
                        {
                            strBin += '0';
                            indexs.Add(i + 1);
                        }
                        else if (shiftR == redA0 && greenA0 == shiftG &&
shiftB == blueA0)
                        {
                            strBin += "A";
                            indexs.Add(i + 1);
                        }
                        else if (shiftR == redA1 && greenA1 == shiftG &&
shiftB == blueA1)
                        {
                            strBin += "B";
                            indexs.Add(i + 1);
                        }
                    }
                }

            if (indexs.Count == 0) indexs = null;

            return strBin;
        }

        #endregion

        #region дополнительные функции

        public Color ToColor(Word.WdColor originalColor)
        {
            int wordColor = (int)originalColor;
            int alphaTemp = (int)((wordColor & 0xFF000000) >> 24);
            int alpha = (alphaTemp == 0) ? 255 : alphaTemp;

            int r = (wordColor & 0x000000FF);
            int g = ((wordColor & 0x0000FF00) >> 8);
            int b = ((wordColor & 0x00FF0000) >> 16);

            return Color.FromArgb(alpha, r, g, b);
        }

        #endregion

        #region Управление настройками


        #region Управления кнопками в меню программы
        private void toolStripButtonSaveSet_Click(object sender, EventArgs e)
        {
            saveSet();
        }
        private void toolStripButtonOpenSet_Click(object sender, EventArgs e)
        {
```

```
            loadSet();
        }
        private void toolStripButtonSaveTemplate_Click(object sender,
EventArgs e)
        {
            saveTemplate();
        }
        private void toolStripButton2_Click(object sender, EventArgs e)
        {
            loadTemplate();
        }

        #endregion

        /// <summary>
        /// сохранение настроек
        /// </summary>
        private void saveSet()
        {
            SaveFileDialog saveFileDialog = new SaveFileDialog();

            saveFileDialog.Filter = "SWord-ключ|*.sword";
            if (saveFileDialog.ShowDialog() == DialogResult.OK)
            {
                String path = saveFileDialog.FileName;
                System.IO.StreamWriter fout = new
System.IO.StreamWriter(path);
                fout.Write(getSet());
                fout.Close();
            }
        }

        /// <summary>
        /// загрузка настроек
        /// </summary>
        private void loadSet()
        {
            OpenFileDialog openFileDialog = new OpenFileDialog();
            openFileDialog.Filter = "SWord-ключ|*.sword";
            if (openFileDialog.ShowDialog() == DialogResult.OK)
            {
                String path = openFileDialog.FileName;

                System.IO.StreamReader sr = new System.IO.StreamReader(path);
                String key = sr.ReadToEnd();
                sr.Close();


                XmlDocument xmlSet = new XmlDocument();
                xmlSet.LoadXml(key);
                setSet(xmlSet);
            }
        }

        /// <summary>
        /// сохранение шаблона
        /// </summary>
        private void saveTemplate()
        {
            String setKey = getSet();
            String setData = getDataTempalate();

            String template = "<template>" + setKey + setData +
"</template>";
```

```
            SaveFileDialog saveFileDialog = new SaveFileDialog();
            saveFileDialog.Filter = "SWord-шаблон|*.swordt";
            if (saveFileDialog.ShowDialog() == DialogResult.OK)
            {
                String path = saveFileDialog.FileName;
                System.IO.StreamWriter fout = new
System.IO.StreamWriter(path);
                fout.Write(template);
                fout.Close();
            }

        }

        /// <summary>
        /// загрузка шаблона
        /// </summary>
        private void loadTemplate()
        {
            OpenFileDialog openFileDialog = new OpenFileDialog();
            openFileDialog.Filter = "SWord-шаблон|*.swordt";
            if (openFileDialog.ShowDialog() == DialogResult.OK)
            {
                String path = openFileDialog.FileName;

                System.IO.StreamReader sr = new System.IO.StreamReader(path);
                String key = sr.ReadToEnd();
                sr.Close();

                XmlDocument templ = new XmlDocument();
                templ.LoadXml(key);

                XmlDocument xmlSet = new XmlDocument();

xmlSet.LoadXml(templ.GetElementsByTagName("infoSet")[0].OuterXml);
                setSet(xmlSet);

                XmlDocument xmlTemplate = new XmlDocument();

xmlTemplate.LoadXml(templ.GetElementsByTagName("infoData")[0].OuterXml);
                setTemplate(xmlTemplate);
            }
        }


        /// <summary>
        /// собрать настройки
        /// </summary>
        /// <returns>настройки скрытия</returns>
        private String getSet()
        {
            String key = "";

            key += "<infoText encoding='" +
currentEncoding.WindowsCodePage.ToString() + "' print='" +
rPrint.Checked.ToString() + "' />";

            String infoArxiv = "none";
            if (cbArxiv.Checked)
            {
                if (rArxiveKey.Checked) infoArxiv = "key";
                else infoArxiv = "inside";
            }
            key += "<infoArxiv arxiv='" + infoArxiv + "' />";
```

```
            string infoColor = (rbDynamicColor.Checked) ? "dynamic" :
"static";
            string color = "";
            if (rbDynamicColor.Checked)
            {
                color = tfR0.Text + "," + tfG0.Text + "," + tfB0.Text + "," +
tfR1.Text + "," + tfG1.Text + "," + tfB1.Text;
            }
            else
            {
                color = tfSR0.Text + "," + tfSG0.Text + "," + tfSB0.Text +
"," + tfSR1.Text + "," + tfSG1.Text + "," + tfSB1.Text;
            }
            key += "<infoColor type='" + infoColor + "' color='" + color + "'
/>";
            key = "<infoSet>" + key + "</infoSet>";

            return key;
        }

        /// <summary>
        /// установить необходимый режим работы
        /// </summary>
        /// <param name="xmlSet"></param>
        private void setSet(XmlDocument xmlSet)
        {
            XmlNode infoText = xmlSet.GetElementsByTagName("infoText")[0];

            currentEncoding =
Encoding.GetEncoding(Convert.ToInt16(infoText.Attributes["encoding"].Value));
            rPrint.Checked = (infoText.Attributes["print"].Value == "True");

            XmlNode infoArxiv = xmlSet.GetElementsByTagName("infoArxiv")[0];
            string arxiv = infoArxiv.Attributes["arxiv"].Value;

            if (arxiv == "none") cbArxiv.Checked = false;
            else
            {
                cbArxiv.Checked = true;

                if (arxiv == "key") rArxiveKey.Checked = true;
                else rArxivInside.Checked = true;
            }

            XmlNode infoColor = xmlSet.GetElementsByTagName("infoColor")[0];
            string color = infoColor.Attributes["type"].Value;

            List <TextBox> tfs = new List<TextBox>();
            if (color == "dynamic")
            {
                rbDynamicColor.Checked = true;
                tfs.Add(tfR0);
                tfs.Add(tfG0);
                tfs.Add(tfB0);
                tfs.Add(tfR1);
                tfs.Add(tfG1);
                tfs.Add(tfB1);
            }
            else
            {
                rbStaticColor.Checked = true;
                tfs.Add(tfSR0);
                tfs.Add(tfSG0);
                tfs.Add(tfSB0);
                tfs.Add(tfSR1);
```

```
                tfs.Add(tfSG1);
                tfs.Add(tfSB1);
            }

            string[] colors = infoColor.Attributes["color"].Value.Split(',');

            for (int i=0;i<colors.Length;i++)
            {
                tfs[i].Text = colors[i];
            }
        }
        /// <summary>
        /// собрать данные для шаблона
        /// </summary>
        /// <returns>собранные данные</returns>
        private string getDataTempalate()
        {
            String data = "";

                data = "<originalText><![CDATA[" + secretText +
"]]></originalText>";
                data += "<arxiv use='" + cbArxiv.Checked + "' key='" +
strKeyArxiv + "'>" + binArxiv + "</arxiv>";
                data = "<infoData>" + data + "</infoData>";

            return data;
        }

        /// <summary>
        /// установить данные в шаблон
        /// </summary>
        /// <param name="xmlTemplate">xml-данные для настройки
шаблона</param>
        private void setTemplate(XmlDocument xmlTemplate)
        {
            String original =
xmlTemplate.GetElementsByTagName("originalText")[0].InnerText;
            secretText = original;

            XmlNode arxiv = xmlTemplate.GetElementsByTagName("arxiv")[0];
            if (arxiv.Attributes["use"].Value != "none")
            {
                strKeyArxiv = arxiv.Attributes["key"].Value;
                binArxiv = arxiv.InnerText;
                profit = countProfit(binArxiv);
                showArxivEffect();
                isChangeText = false;
            }
        }

        #endregion

        /// <summary>
        /// Настройка для тестирования работы программы
        /// </summary>
        private void tempStart()
        {
            toolStripComboBox1.SelectedIndex = 1;
            secretText = "привет, ";
            tfB0.Text = "0";
            tfB1.Text = "1";
            tfG0.Text = "1";
            tfG1.Text = "0";
            tfR0.Text = "0";
            tfR1.Text = "0";
```

```
            tfAB0.Text = "0";
            tfAB1.Text = "2";
            tfAG0.Text = "2";
            tfAG1.Text = "0";
            tfAR0.Text = "0";
            tfAR1.Text = "0";
        }
    }
}

using System;
using System.Collections.Generic;
using System.Text;

namespace sword
{
    class BY
    {
        public static String getBY(String plainText)
        {

           String mes = plainText;
           string word = "";

           List<String> astr = new List<String>();

           int ml = mes.Length;

           for (int i = 0; i < ml; i++) astr.Insert(0, mes.Substring(i, ml -
i) + mes.Substring(0, i));

           astr.Sort();
           int N = -1;
           for (int i = 0; i < ml && N == -1; i++) if (astr[i] == mes) N = i;
           word = "";
           for (int i = 0; i < ml; i++) word += astr[i][ml - 1];
           word += "|" + N;

            return word;
        }

        public static String extractBY(String plainText, int num)
        {

            String mes = plainText;
            List<String> astr = new List<String>();
            int ml = mes.Length;

            for (int i = 0; i < ml; i++) astr.Insert(0,"");

            for (int i = 0; i < ml; i++){
                for (int q = 0; q < ml; q++)
                    astr[q] = mes[q]+astr[q];

                astr.Sort();
            }
            astr.Sort();
            return astr[num];
         }
    }
}
```