Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский университет

«Московский институт электронной техники»

институт системной и программной инженерии

и информационных технологий»

Курс «Теория алгоритмических языков и компиляторов»

Лабораторная работа № 6

Разработка процессора языка разметки документа.

Выполнил:

Студент группы ПИН-51Д

Джугели Дмитрий Александрович

Москва, 2025

Задание на лабораторную работу

Написать процессор, обрабатывающий документы на языке eMark.

Входные данные: текстовый файл, содержащий документ на языке eMark.

Выходные данные: консоль с отформатированным текстом в соответствии с управляющими инструкциями, либо с сообщениями об ошибках в документе

Код:

Класс Program.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Threading;
using System.Xml;
using System.Xml.Serialization;

namespace Emark
{
    public class Program
    {
        private static MainWindow _window;
        private const string filename = "items.xml";
        private XmlSerializer _formatter = new XmlSerializer(typeof(Block));

        static void Main(string[] args)
        {
            var windowThred = new Thread(() =>
            {
                _window = new MainWindow(Update);
                var block = new Block
                {
                    Rows = 1,
                    Columns = 2,
                    Column = new RowOrColumn[]
                    {
                        new RowOrColumn(){ VAlign = VAlign.Bottom },
                        new RowOrColumn(){ VAlign = VAlign.Center },
                    },
                };
                _window.Show();
                _window.Closed += (s, e) =>
_window.Dispatcher.BeginInvokeShutdown(DispatcherPriority.Normal);
                Dispatcher.Run();
            });
            windowThred.SetApartmentState(ApartmentState.STA);
            windowThred.Start();
        }

        private static void Update(string text)
        {
            new Task(() =>
            {
                try
                {
                    Console.CursorSize = 1;
                    Console.Clear();
                    var reader = new XmlTextReader(new StringReader(text))
                    {
                        WhitespaceHandling = WhitespaceHandling.None,
                    };
                    var element = CreateElement(text.Deserialize());
                    WriteBlock(element);
                    Console.SetCursorPosition(0, 25);
                    Console.BackgroundColor = ConsoleColor.Black;
```

```
                    Console.Write(' ');
                    _window.OnError(null);
                }
                catch (Exception ex)
                {
                    _window.OnError($"{ex.Message} ({ex.InnerException?.Message})");
                }
            }).Start();
        }

        private static void WriteBlock(Element element)
        {
            if (element is Text text)
            {
                PaintBackground(element);
                var countRows = text.Value.Length / element.Width;
                countRows += text.Value.Length % element.Width > 0 ? 1 : 0;

                var top = element.VAlign == VAlign.Top
                    ? 0
                    : element.VAlign == VAlign.Center
                    ? (element.Heigth - countRows) / 2
                    : element.Heigth - countRows;

                var i = 0;
                for (; i + element.Width < text.Value.Length && top < element.Heigth; i +=
element.Width, top++)
                {
                    Console.SetCursorPosition(element.X, element.Y + top);
                    Console.Write(text.Value.Substring(i, Math.Min(element.Width,
text.Value.Length - i)));
                }
                var currentText = text.Value.Substring(i, Math.Min(element.Width,
text.Value.Length - i));
                var left = element.HAlign == HAlign.Left
                    ? element.X
                    : element.HAlign == HAlign.Center
                    ? element.X + (element.Width - currentText.Length) / 2
                    : element.X + element.Width - currentText.Length;
                Console.SetCursorPosition(left, element.Y + top);
                Console.Write(currentText);
            }
            else if (element is Panel panel)
            {
                foreach (var child in panel.Children)
                    WriteBlock(child);
            }
        }
private static void PaintBackground(Element element)
        {
            Console.BackgroundColor = element.Background;
            Console.ForegroundColor = element.Foreground;
            for (var i = element.X; i < element.X + element.Width; i++)
                for (var j = element.Y; j < element.Y + element.Heigth; j++)
                {
                    Console.SetCursorPosition(i, j);
                    Console.Write(' ');
                }
        }

        private static Element CreateElement(Block block)
        {
            return BlockOrTextParse(block, 80, 24, 0, 0, ConsoleColor.Black,
ConsoleColor.White, HAlign.Left, VAlign.Top, true);
        }
```

```
        private static void BaseBlockTest(BaseBlock block)
        {
            var count = 0;
            void Inc(object value) { if (value != null) count++; }
            Inc(block.Column);
            Inc(block.Row);
            Inc(block.Value);
            if (count > 1)
                throw new Exception("Можно указать либо строки, либо колонки, либо занчение");
        }

        private static Element BlockOrTextParse(BaseBlock baseBlock, int width, int height,
int x, int y, ConsoleColor background, ConsoleColor foreground, HAlign hAlign, VAlign vAlign,
bool isFirst = false)
        {
            BaseBlockTest(baseBlock);

            if (baseBlock is Block block)
            {
                if (block.Rows == null)
                {
                    if (!isFirst)
                        throw new Exception("Не указано количество строк у block");
                    block.Rows = 0;
                }
                if (block.Columns == null)
                {
                    if (!isFirst)
                        throw new Exception("Не указано количество столбцов у block");
                    block.Columns = 1;
                }

                if (baseBlock.Column != null)
                {
                    var columnsCount = block.Columns.Value;
                    if (baseBlock.Column.Length != columnsCount)
                        throw new Exception($"У block требуемое количество столбцов:
{columnsCount}");

                    var children = new Element[columnsCount, block.Rows.Value > 0 ?
block.Rows.Value : 1];
                    var left = 0;
                    for (var i = 0; i < columnsCount; i++)
                    {
                        if (i < columnsCount - 1 && baseBlock.Column[i].Width == null)
                            throw new Exception("Значение width должны быть указаны у всех
колонок, кроме последней");
                        var columnWidth = Math.Min(baseBlock.Column[i].Width ?? width - left,
width - left);

                        var row = RowsParse(baseBlock.Column[i],
                            block.Rows.Value,
                            columnWidth,
                            height,
                            x + left,
                            y,
                            baseBlock.Column[i].Background ?? background,
                            baseBlock.Column[i].Foreground ?? foreground,
                            baseBlock.Column[i].HAlign ?? hAlign,
                            baseBlock.Column[i].VAlign ?? vAlign);

                        left += columnWidth;
                        for (var j = 0; j < row.Length; j++)
                            children[i, j] = row[j];
```

```
                    }

                    return new Panel(children, width, height, x, y, background, foreground,
hAlign, vAlign);
                }
                else if (baseBlock.Row != null)
                {
                    var rowsCount = block.Rows.Value;
                    if (baseBlock.Row.Length != rowsCount)
                        throw new Exception($"У block требуемое количество строк:
{rowsCount}");

                    var children = new Element[block.Columns.Value > 0 ? block.Columns.Value :
1, rowsCount];
                    var top = 0;
                    for (var i = 0; i < rowsCount; i++)
                    {
                        if (i < rowsCount - 1 && baseBlock.Row[i].Height == null)
                            throw new Exception("Значение height должны быть указаны у всех
строк, кроме последней");
                        var rowHeight = Math.Min(baseBlock.Row[i].Height ?? height - top,
height - top);

                        var row = ColumnsParse(baseBlock.Row[i],
                            block.Columns.Value,
                            width,
                            rowHeight,
                            x,
                            y + top,
                            baseBlock.Row[i].Background ?? background,
                            baseBlock.Row[i].Foreground ?? foreground,
                            baseBlock.Row[i].HAlign ?? hAlign,
                            baseBlock.Row[i].VAlign ?? vAlign);
                        top += rowHeight;
                        for (var j = 0; j < row.Length; j++)
                            children[j, i] = row[j];
                    }

                    return new Panel(children, width, height, x, y, background, foreground,
hAlign, vAlign);
                }
                else
                {
                    if (block.Columns.Value != 0)
                        throw new Exception($"Ожидаемое количество колонок:
{block.Columns.Value}");
                    if (block.Rows.Value != 0)
                        throw new Exception($"Ожидаемое количество строк:
{block.Rows.Value}");
                    return new Text(baseBlock.Value ?? "", width, height, x, y, background,
foreground, hAlign, vAlign);
                }
            }
            else if (isFirst)
                throw new Exception("Ожидается block");

            if (baseBlock.Column != null)
            {
                throw new Exception($"Ожидается block, либо текст");
            }
            else if (baseBlock.Column != null)
            {
                throw new Exception($"Ожидается block, либо текст");
            }
            else if (baseBlock.Value != null)
```

```
            {
                return new Text(baseBlock.Value, width, height, x, y, background, foreground,
hAlign, vAlign);
            }
            else if (baseBlock is RowOrColumn rowOrColumn && rowOrColumn.Block != null)
                return BlockOrTextParse(rowOrColumn.Block, width, height, x, y, background,
foreground, hAlign, vAlign);

            return new Text("", width, height, x, y, background, foreground, hAlign, vAlign);
        }
  public static Element[] RowsParse(RowOrColumn column, int countRows, int width, int height,
int x, int y, ConsoleColor background, ConsoleColor foreground, HAlign hAlign, VAlign vAlign)
        {
            if (countRows == 0)
            {
                if (column.Column != null && column.Column.Length > 0 || column.Row != null &&
column.Row.Length > 0)
                    throw new Exception("Колонки и строки не ожидаются");
                return new Element[] { new Text(column.Value ?? "", width, height, 0, 0,
background, foreground, hAlign, vAlign) };
            }

            if (countRows > 0 && (column.Row == null || column.Row.Length != countRows))
                throw new Exception($"Ожидаемое количество строк: {countRows}");
            if (column.Column != null)
                throw new Exception($"Ожидаемое количество строк: {countRows}. Колонки не
ожидаются");

            var top = 0;
            var elements = new Element[countRows];
            for (var i = 0; i < countRows; i++)
            {
                if (i < countRows - 1 && column.Row[i].Height == null)
                    throw new Exception("Значение height должны быть указаны у всех строк,
кроме последней");
                var rowHeight = Math.Min(column.Row[i].Height ?? height - top, height - top);

                elements[i] = BlockOrTextParse(column.Row[i],
                    width,
                    rowHeight,
                    x,
                    y + top,
                    column.Row[i].Background ?? background,
                    column.Row[i].Foreground ?? foreground,
                    column.Row[i].HAlign ?? hAlign,
                    column.Row[i].VAlign ?? vAlign);
                top += rowHeight;
            }

            return elements;
        }
public static Element[] ColumnsParse(RowOrColumn row, int countColumns, int width, int height,
int x, int y, ConsoleColor background, ConsoleColor foreground, HAlign hAlign, VAlign vAlign)
        {
            if (countColumns == 0)
            {
                if (row.Column != null && row.Column.Length > 0 || row.Row != null &&
row.Row.Length > 0)
                    throw new Exception("Колонки и строки не ожидаются");
                return new Element[] { new Text(row.Value ?? "", width, height, x, y,
background, foreground, hAlign, vAlign) };
            }

            if (countColumns > 0 && (row.Column == null || row.Column.Length != countColumns))
                throw new Exception($"Ожидаемое количество колонок: {countColumns}");
```

```
            if (row.Row != null)
                throw new Exception($"Ожидаемое количество колонок: {countColumns}. Строки не
ожидаются");


            var elements = new Element[countColumns];
            var left = 0;
            for (var i = 0; i < countColumns; i++)
            {
                if (i < countColumns - 1 && row.Column[i].Width == null)
                    throw new Exception("Значение width должны быть указаны у всех колонок,
кроме последней");
                var columnWidth = Math.Min(row.Column[i].Width ?? width - left, width - left);

                elements[i] = BlockOrTextParse(row.Column[i], columnWidth, height, x + left,
y,
                    row.Column[i].Background ?? background,
                    row.Column[i].Foreground ?? foreground,
                    row.Column[i].HAlign ?? hAlign,
                    row.Column[i].VAlign ?? vAlign);
                left += columnWidth;
            }

            return elements;
        }
    }

    [Serializable]
    public enum VAlign
    {
        [XmlEnum("top")]
        Top,
        [XmlEnum("center")]
        Center,
        [XmlEnum("bottom")]
        Bottom
    }

    public enum HAlign
    {
        [XmlEnum("left")]
        Left,
        [XmlEnum("center")]
        Center,
        [XmlEnum("right")]
        Right
    }

    [Serializable]
    public class BaseBlock
    {
        [XmlElement("column")]
        public RowOrColumn[] Column;
        [XmlElement("row")]
        public RowOrColumn[] Row;
        [XmlText]
        public string Value;
    }

    [XmlRoot("block")]
    public class Block : BaseBlock
    {
        [XmlIgnore]
        public int? Rows;
        [XmlIgnore]
```

```
        public int? Columns;

        [XmlAttribute("rows")]
        [EditorBrowsable(EditorBrowsableState.Never)]
        public string RowsValue { get => Rows.Value.ToString(); set => Rows =
(int)uint.Parse(value); }
        [XmlAttribute("columns")]
        [EditorBrowsable(EditorBrowsableState.Never)]
        public string ColumnsValue { get => Columns.Value.ToString(); set => Columns =
(int)uint.Parse(value); }
    }

    [Serializable]
public class RowOrColumn : BaseBlock
    {
        [XmlIgnore]
        public VAlign? VAlign;
        [XmlIgnore]
        public HAlign? HAlign;
        [XmlIgnore]
        public ConsoleColor? Foreground;
        [XmlIgnore]
        public ConsoleColor? Background;
        [XmlIgnore]
        public int? Width;
        [XmlIgnore]
        public int? Height;

        [XmlElement("block")]
        public Block Block;

        [XmlAttribute("textcolor")]
        [EditorBrowsable(EditorBrowsableState.Never)]
        public string TextColor { get => Foreground.Value.ToString(); set => Foreground =
value.ConsoleColorParse(); }
        [XmlAttribute("bgcolor")]
        [EditorBrowsable(EditorBrowsableState.Never)]
        public string Bgcolor { get => Background.Value.ToString(); set => Background =
value.ConsoleColorParse(); }
        [XmlAttribute("valign")]
        [EditorBrowsable(EditorBrowsableState.Never)]
        public string VAlignValue { get => VAlign.Value.ToString(); set { VAlign =
value.VAlignParse(); } }
        [XmlAttribute("halign")]
        [EditorBrowsable(EditorBrowsableState.Never)]
        public string HAlignValue { get => HAlign.Value.ToString(); set { HAlign =
value.HAlignParse(); } }
        [XmlAttribute("width")]
        [EditorBrowsable(EditorBrowsableState.Never)]
        public string WidthValue { get => Width.Value.ToString(); set { Width =
(int)uint.Parse(value); } }
        [XmlAttribute("height")]
        [EditorBrowsable(EditorBrowsableState.Never)]
        public string HeightValue { get => Height.Value.ToString(); set { Height =
(int)uint.Parse(value); } }
    }
public abstract class Element
    {
        public readonly int Width;
        public readonly int Heigth;
        public readonly ConsoleColor Background;
        public readonly ConsoleColor Foreground;
        public readonly HAlign HAlign;
        public readonly VAlign VAlign;
        private readonly int _x;
```

```
        private readonly int _y;
        private Panel _parent;

        public Element(int width, int height, int x, int y, ConsoleColor background,
ConsoleColor foreground, HAlign hAlign, VAlign vAlign)
        {
            Width = width;
            Heigth = height;
            _x = x;
            _y = y;
            Background = background;
            Foreground = foreground;
            HAlign = hAlign;
            VAlign = vAlign;
        }

        public int X => _x;
        public int Y => _y;


        public void SetParent(Panel parent)
        {
            _parent = parent;
        }
    }

    public class Panel : Element
    {
        public readonly Element[,] Children;

        public Panel(Element[,] children, int width, int height, int x, int y, ConsoleColor
background, ConsoleColor foreground, HAlign hAlign, VAlign vAlign)
            : base(width, height, x, y, background, foreground, hAlign, vAlign)
        {
            Children = children;
            foreach (var child in children)
            {
                child.SetParent(this);
            }
        }
    }

    public class Text : Element
    {
        private static IReadOnlyCollection<char> _spaces = new HashSet<char>()
        {
            '\t', '\r', '\n', ' '
        };
        public readonly string Value;

        public Text(string value, int width, int height, int x, int y, ConsoleColor
background, ConsoleColor foreground, HAlign hAlign, VAlign vAlign)
            : base(width, height, x, y, background, foreground, hAlign, vAlign)
        {
            var start = 0;
            for (start = 0; start < value.Length; start++)
                if (!_spaces.Contains(value[start]))
                    break;
            var end = 0;
            for (end = value.Length - 1; end > 0; end--)
                if (!_spaces.Contains(value[end]))
                    break;
            Value = value.Substring(start, end - start + 1);
        }
    }
```

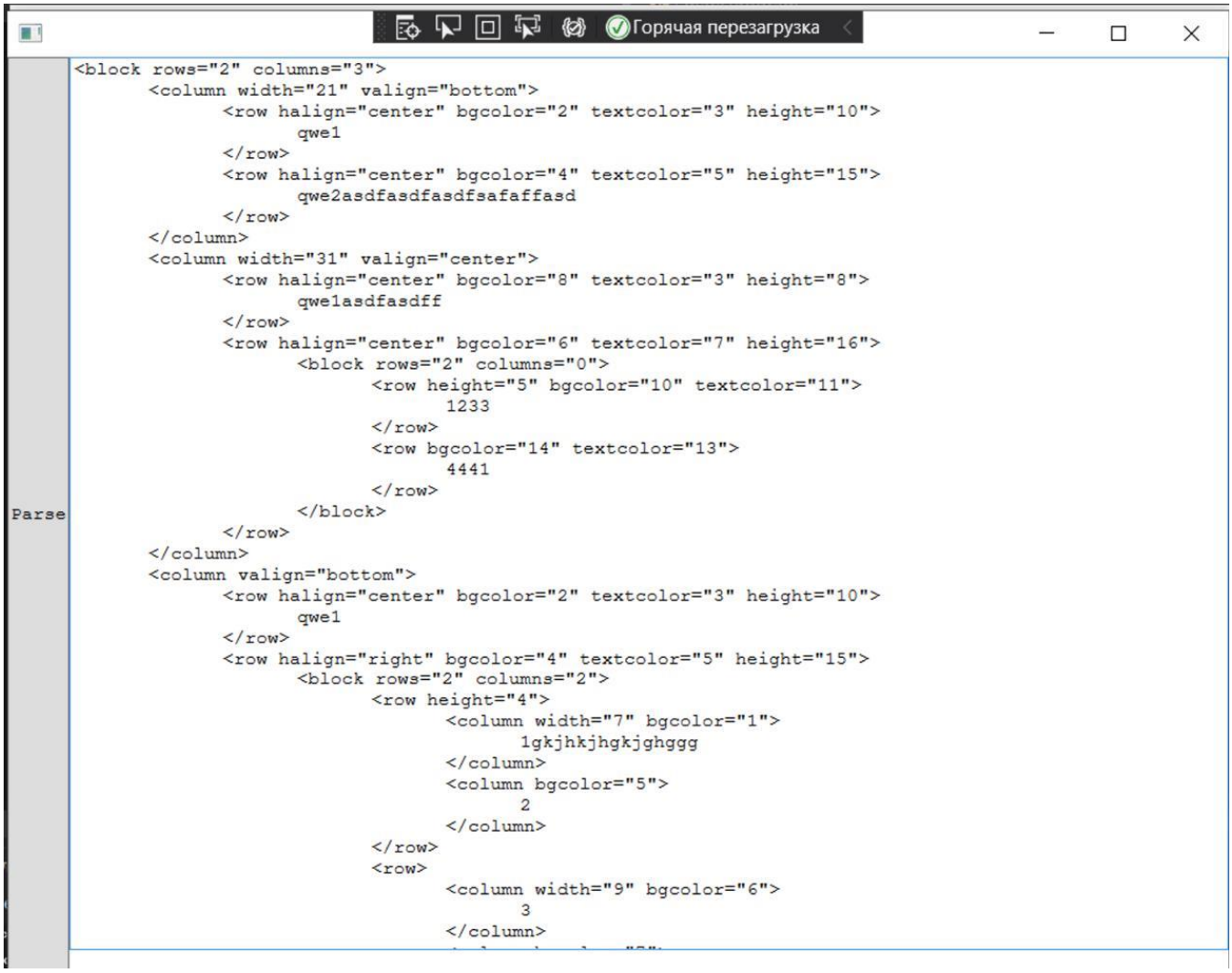
```
    public static class Extensions
    {
        public static void CreateException(this XmlTextReader reader, string message)
        {
            throw new Exception($"[{reader.LineNumber} {reader.LinePosition}] {message}");
        }

        private static readonly XmlSerializer _serializer = new XmlSerializer(typeof(Block));

        public static string SerializeObject(this Block toSerialize)
        {
            using (StringWriter textWriter = new StringWriter())
            {
                _serializer.Serialize(textWriter, toSerialize);
                return textWriter.ToString();
            }
        }

        public static Block Deserialize(this string text)
        {
            return (Block)_serializer.Deserialize(new StringReader(text));
        }

        private static readonly Type _consoleColorType = typeof(ConsoleColor);
        private static readonly Type _hAlignType = typeof(HAlign);
        private static readonly Type _valignType = typeof(VAlign);

        public static ConsoleColor ConsoleColorParse(this string value)
        {
            return (ConsoleColor)Enum.Parse(_consoleColorType, value, true);
        }

        public static HAlign HAlignParse(this string value)
        {
            return (HAlign)Enum.Parse(_hAlignType, value, true);
        }

        public static VAlign VAlignParse(this string value)
        {
            return (VAlign)Enum.Parse(_valignType, value, true);
        }
    }
}
```

Класс MainWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
```

```
namespace Emark
{
    public partial class MainWindow : Window
    {
        private readonly Action<string> _textChaged;

        public MainWindow(Action<string> textChanged)
        {
            InitializeComponent();

            _textChaged = textChanged;
            Program.Text = File.ReadAllText("file.txt");
            Closing += (s, e) => File.WriteAllText("file.txt", Program.Text);
            Program.TextChanged += Program_TextChanged;
        }

        private void Program_TextChanged(object sender, TextChangedEventArgs e)
        {
            _textChaged(Program.Text);
        }

        public void OnError(string error)
        {
            Dispatcher.BeginInvoke((Action)(() =>
            {
                Error.Text = error;
            }));
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            _textChaged(Program.Text);
        }
    }
}
```

Примеры работы:




```
<block rows="2" columns="3">
	<column width="21" valign="bottom">
		<row halign="center" bgcolor="2" textcolor="3" height="10">
			qwe1
		</row>
		<row halign="center" bgcolor="4" textcolor="5" height="15">
			qwe2asdfasdfasdfsafaffasd
		</row>
	</column>
	<column width="31" valign="center">
		<row halign="center" bgcolor="8" textcolor="3" height="8">
			qwe1asdfasdff
		</row>
		<row halign="center" bgcolor="6" textcolor="7" height="16">
			<block rows="2" columns="0">
				<row height="5" bgcolor="10" textcolor="11">
					1233
				</row>
				<row bgcolor="14" textcolor="13">
					4441
				</row>
			</block>
		</row>
	</column>
	<column valign="bottom">
		<row halign="center" bgcolor="2" textcolor="3" height="10">
			qwe1
		</row>
		<row halign="right" bgcolor="4" textcolor="5" height="15">
			<block rows="2" columns="2">
				<row height="4">
					<column width="7" bgcolor="1">
						1gkjhkjhgkjghggg
					</column>
					<column bgcolor="5">
						2
					</column>
				</row>
				<row>
					<column width="9" bgcolor="6">
						3
					</column>
```