```
class Schema
{
    private string pattern = @"^\s*(?<redefines>R?)(?<level>\d+)\s+(?<varName>\S+)((\s+(?<type>[xcnpXCNP])\s+((?<length>\d+))(\,(?<decimalPlaces>\d+))?)?(\s+(?<repeatCount>\d+))?)?(\s{2,}(?<comment>.*))?$";
    private string schemaStr = "";

    public AbstractNode ParseLine(string line)
    {
        // gibt eine Value- oder GroupNode zurück, je nachdem ob line
        // Angaben zu Typ und ByteAnzahl hat oder nicht
        // wenn pattern nicht matcht wird null zurückgegeben
    }

    public GroupNode Parse()
    {
        var stack = new Stack<AbstractNode>();
        var root = new GroupNode(false, 0, "root", 1, 1, "");
        stack.Push(root);

        Action addChildFromStackToParent =
            () =>
            {
                var child = stack.Pop();
                var parent = stack.Peek();
                parent.AddChild(child);
                for (int currentRepeatIndex = 2; currentRepeatIndex <= child.RepeatCount; ++currentRepeatIndex)
                {
                    parent.AddChild(child.CreateCopyWithIndex(currentRepeatIndex));
                }

            };

        var schemaLines = schemaStr.Split(new string[] { "\r\n", "\n" }, StringSplitOptions.RemoveEmptyEntries);
        foreach (var currentLine in schemaLines)
        {
            var currentNode = ParseLine(currentLine);
            if (currentNode == null) { continue; }
            while (currentNode.Level <= stack.Peek().Level)
            {
                addChildFromStackToParent();
            }
            stack.Push(currentNode);
        }
        while (stack.Count >= 2)
        {
            addChildFromStackToParent();
        }
        return root;
    }
}


```

```csharp
class GroupNode : AbstractNode
{
    private List<AbstractNode> children = new List<AbstractNode>();
    public GroupNode(bool redefines, int level, string varName, int repeatCount, int repeatIndex, string comment)
        : base(redefines, level, varName, repeatCount, repeatIndex, comment)
    { }

    public override int AssignValue(string data)
    {
        int currentShift = 0;
        int totalShift = 0;
        foreach (var child in children)
        {
            if (!child.Redefines)
            {
                currentShift = child.AssignValue(data.Substring(totalShift));
                totalShift += currentShift;
            }
            else
            {
                child.AssignValue(data.Substring(totalShift - currentShift));
            }
        }
        return totalShift;
    }

    public override string ToString(int tabCount)
    {
        StringBuilder strBuilder = new StringBuilder();
        if (Level != 0)
        {
            strBuilder.Append(string.Format("{0}{1}{2} {3}{4}\r\n",
            new string(' ', tabCount * 4 - (Redefines ? 1 : 0)),
            Redefines ? "R" : "",
            Level.ToString().PadLeft(2, '0'),
            VarName,
            RepeatCount > 1 ? string.Format("({0})", RepeatIndex) : ""));
        }
        foreach (var child in children)
        {
            strBuilder.Append(child.ToString(tabCount + (Level == 0 ? 0 : 1)));
        }
        return strBuilder.ToString();
    }

    public override AbstractNode CreateCopyWithIndex(int index)
    {
        GroupNode g = new GroupNode(Redefines, Level, VarName, RepeatCount, index, Comment);
        foreach (var child in children)
        {
            g.AddChild(child.CreateCopyWithIndex(child.RepeatIndex));
        }
        return g;
    }
}


class ValueNode : AbstractNode
{
    public string Type { get; private set; }
    public int Length { get; private set; }
    public string Value { get; private set; }

    public ValueNode(bool redefines, int level, string varName, string type,
      int length, int repeatCount, int repeatIndex, string comment)
        : base(redefines, level, varName, repeatCount, repeatIndex, comment)
    {
        Type = type;
        Length = length;
    }

    public override AbstractNode CreateCopyWithIndex(int index)
    {
        return new ValueNode(Redefines, Level, VarName, Type, Length,
          RepeatCount, index, Comment);
    }

    public override int AssignValue(string data)
    {
        Value = data.Substring(0, Length);
        return Length;
    }

    public override string ToString(int tabCount)
    {
        if (Level == 0)
        {
            return "";
        }
        return string.Format("{0}{1}{2} {3}{4}={5}\r\n",
            new string(' ', tabCount * 4 - (Redefines ? 1 : 0)),
            Redefines ? "R" : "",
            Level.ToString().PadLeft(2, '0'),
            VarName,
            (RepeatCount > 1) ? ("(" + RepeatIndex + ")") : "",
            Value);
    }

    public override void AddChild(AbstractNode child)
    {
        throw new InvalidOperationException(this.VarName + ": " + "Werteknoten
           können keine Kindknoten haben!");
    }
}

```