```
<!DOCTYPE html>
<html>
<head>
      <meta charset="UTF-8">
      <title>Исходный код программы</title>
</head>
<body>
      <h1>Исходный код программы</h1>
      <p><pre>
class Parser
    {
        public struct ParsingError
        {
            public enum ActionOverItem
            {
                Remove,
                Replace,
                InsertAfter,
                InsertBefore,
            }
            public int position { get; set; }
            public LexicalScanner.Codes expectedItem { get; set; }
            public ActionOverItem action { get; set; }
            public string message { get; set; }

            public ParsingError(LexicalScanner.Codes expectedItem,
ActionOverItem action, int position, string message)
            {
                this.expectedItem = expectedItem;
                this.action = action;
                this.position = position;
                this.message = message;
            }
        }
        private static List<LexicalItem> s_tokens;
        private static int s_currentTokenIndex;
        private static bool s_IsBoolIdentidier;
        public static List<ParsingError> s_errors { get; private set; }

        public static void ClearErrorsList() { s_errors.Clear(); }

        // Метод для проверки текущей лексемы и перехода к следующей
        private static bool Check(LexicalScanner.Codes expectedCode)
        {
            if (s_currentTokenIndex == s_tokens.Count)
            {
                return false;
            }
            return s_tokens[s_currentTokenIndex].lexicalCode ==
expectedCode;
        }

        private static void Match(LexicalScanner.Codes expectedCode)
        {
            if (s_currentTokenIndex < s_tokens.Count)
            {
                if (s_tokens[s_currentTokenIndex].lexicalCode ==
expectedCode)
                {
                    s_currentTokenIndex++;
                }
                else
                {
                    if (!Check(LexicalScanner.Codes.ErrorCode) &&
!Check(LexicalScanner.Codes.RightParenCode))
                    {
                        var error = new ParsingError(expectedCode,
ParsingError.ActionOverItem.InsertBefore,
s_tokens[s_currentTokenIndex].startPosition - 1, $"Ожидалось:
{expectedCode}, получен {s_tokens[s_currentTokenIndex].lexicalCode}");
                        s_errors.Add(error);
                    }
                    else
                    {
                        var error = new ParsingError(expectedCode,
ParsingError.ActionOverItem.InsertBefore,
s_tokens[s_currentTokenIndex].startPosition, $"Ожидалось: {expectedCode},
получен {s_tokens[s_currentTokenIndex].lexicalCode}");
                        s_errors.Add(error);
                    }
                    throw new Exception($"Ожидалась лексема
{expectedCode}, получена {s_tokens[s_currentTokenIndex].lexicalCode}");
                }
            }
        }

        private static bool IsLiteral(int i)
        {
            return (s_tokens[s_currentTokenIndex + i].lexicalCode ==
Codes.IdentifierCode
                || s_tokens[s_currentTokenIndex + i].lexicalCode ==
Codes.LeftParenCode
                || s_tokens[s_currentTokenIndex + i].lexicalCode ==
Codes.DoubleConstCode
                || s_tokens[s_currentTokenIndex + i].lexicalCode ==
Codes.IntegerConstCode);
        }

        public static void ParseInit(List<LexicalItem> inputTokens)
        {
            s_tokens = inputTokens;
            s_currentTokenIndex = 0;
            s_errors = new List<ParsingError>();
            Parse();
        }

        public static void Parse()
        {
            try
            {
                OrExpr();
                if (s_currentTokenIndex != s_tokens.Count)
                {
                    string message = "Некорректный токен";
                    if (s_currentTokenIndex == s_tokens.Count - 1)
                    {
                        var error = new
ParsingError(LexicalScanner.Codes.ErrorCode,
ParsingError.ActionOverItem.Remove,
s_tokens[s_currentTokenIndex].startPosition, message);
                        s_errors.Add(error);
                        throw new Exception(message);
                    }
                    else
                    {
                        if (IsLiteral(0))
                        {
                            message = "Ожидался оператор сравнения";
                            var error = new
ParsingError(Codes.RelationalOpCode,
ParsingError.ActionOverItem.InsertBefore,
s_tokens[s_currentTokenIndex].startPosition - 1, message);
                            s_errors.Add(error);
                            throw new Exception(message);
                        }
                        else if (IsLiteral(1))
                        {
                            message = "Ожидался оператор сравнения";
                            if (!Check(Codes.ErrorCode) &&
!Check(Codes.RightParenCode))
                            {
                                var error = new
ParsingError(Codes.RelationalOpCode,
ParsingError.ActionOverItem.InsertBefore,
s_tokens[s_currentTokenIndex].startPosition - 1, message);
                                s_errors.Add(error);
                            }
                            else
                            {
                                var error = new
ParsingError(Codes.RelationalOpCode, ParsingError.ActionOverItem.Replace,
s_tokens[s_currentTokenIndex].startPosition, message);
                                s_errors.Add(error);
                            }
                            throw new Exception(message);
                        }
                        else
                        {
                            var error = new ParsingError(Codes.ErrorCode,
ParsingError.ActionOverItem.Remove,
s_tokens[s_currentTokenIndex].startPosition, message);
```

```
                            s_errors.Add(error);
                            throw new Exception(message);
                        }
                    }
                }
            }
            catch //Нейтрализация ошибки
            {
                var token = new LexicalItem(s_errors[s_errors.Count - 
1].expectedItem, " ", s_errors[s_errors.Count - 1].position, 
s_errors[s_errors.Count - 1].position);
                if (s_tokens.Count > 1)
                {
                    if (s_currentTokenIndex < s_tokens.Count)
                    {
                        switch (s_errors[s_errors.Count - 1].action)
                        {
                            case 
ParsingError.ActionOverItem.InsertBefore:
                                s_tokens.Insert(s_currentTokenIndex, 
token);

                                break;
                            case ParsingError.ActionOverItem.InsertAfter:
                                s_tokens.Insert(s_currentTokenIndex, 
token);

                                break;
                            case ParsingError.ActionOverItem.Replace:
                                s_tokens.RemoveAt(s_currentTokenIndex);
                                s_tokens.Insert(s_currentTokenIndex, 
token);

                                break;
                            case ParsingError.ActionOverItem.Remove:
                                s_tokens.RemoveAt(s_currentTokenIndex);
                                break;
                        }
                    }
                    else
                    {
                        s_tokens.Add(token);
                    }
                    s_currentTokenIndex = 0;
                    Parse();
                }
                else
                {
                    s_tokens.Remove(token);
                    return;
                }
            }
        }

        // <RelExpr> -> <AddExpr>(RealOp <AddExpr>)*
        private static void OrExpr()
        {
```

```
        AndExpr();
        while ((s_currentTokenIndex < s_tokens.Count)
            && Check(LexicalScanner.Codes.LogicalOpCode)
            && s_tokens[s_currentTokenIndex].item.ToLower() == "or")
        {
            Match(LexicalScanner.Codes.LogicalOpCode);
            AndExpr();
        }
    }

    // <AndExpr> -> <MulExpr> (AddOp <MulExpr>)*
    private static void AndExpr()
    {
        NotExpr();
        while ((s_currentTokenIndex < s_tokens.Count)
            && Check(LexicalScanner.Codes.LogicalOpCode)
            && s_tokens[s_currentTokenIndex].item.ToLower() == "and")
        {
            Match(LexicalScanner.Codes.LogicalOpCode);
            NotExpr();
        }
    }

    // <NotExpr> -> ! <NotExpr> | <RelExpr>
    private static void NotExpr()
    {
        if (Check(LexicalScanner.Codes.NotOpCode))
        {
            s_IsBoolIdentidier = true;
            Match(LexicalScanner.Codes.NotOpCode);
        }
        RelExpr();
    }

    // <RelExpr> -> <AddExpr> (RelOp AddExpr)*
    public static void RelExpr()
    {
        AddExpr();
        while ((s_currentTokenIndex < s_tokens.Count)
            && Check(Codes.LogicalOpCode))
        {
            Match(Codes.LogicalOpCode);
            AddExpr();
        }
    }

    // <AddExpr> -> <MulExpr> (AddOp <MulExpr>)*
    private static void AddExpr()
    {
        MulExpr();                                    //////////////
        while (s_currentTokenIndex < s_tokens.Count
            && Check(Codes.RelationalOpCode))
        {
            Match(LexicalScanner.Codes.RelationalOpCode);
```

```
                MulExpr(); //wer
            }
        }

        // <MulExpr> -> <UnaryExpr>(MulOp <UnaryExpr>)*
        private static void MulExpr()
        {
            UnaryExpr();
            while ((s_currentTokenIndex < s_tokens.Count)
                && Check(LexicalScanner.Codes.AdditiveOpCode))
            {
                Match(LexicalScanner.Codes.AdditiveOpCode);
                UnaryExpr();
            }
        }

        // <UnaryExpr> -> <Factor> | (AddOp <UnaryExpr>)*
        private static void UnaryExpr()
        {
            Factor();
            while ((s_currentTokenIndex < s_tokens.Count)
                && Check(LexicalScanner.Codes.MultiplicateOpCode))
            {
                Match(LexicalScanner.Codes.MultiplicateOpCode);
                Factor();
            }
        }

        // <Factor> -> <Identifier> | Const | '(' Expr ')'
        private static void Factor()
        {
            if (s_currentTokenIndex < s_tokens.Count)
            {
                switch (s_tokens[s_currentTokenIndex].lexicalCode)
                {
                    case LexicalScanner.Codes.LeftParenCode:
                        Match(LexicalScanner.Codes.LeftParenCode);
                        OrExpr();
                        if (!Check(Codes.RightParenCode))
                        {
                            if (s_currentTokenIndex < s_tokens.Count)
                            {
                                if (IsLiteral(0))
                                {
                                    string message = "Ожидался оператор
сравнения";
                                    var error = new
ParsingError(Codes.RelationalOpCode,
ParsingError.ActionOverItem.InsertBefore,
s_tokens[s_currentTokenIndex].startPosition - 1, message);
                                    s_errors.Add(error);
                                    throw new Exception(message);
                                }
                                else if (!Check(Codes.ErrorCode) &&
!Check(Codes.NotOpCode))
                                {
                                    string message = "Ожидалась
закрывающая скобка";
                                    var error = new
ParsingError(Codes.RightParenCode,
ParsingError.ActionOverItem.InsertAfter,
s_tokens[s_currentTokenIndex].endPosition + 1, message);
                                    s_errors.Add(error);
                                    throw new Exception(message);
                                }
                            }
                            else
                            {
                                string message = "Ожидалась закрывающая
скобка";
                                var error = new
ParsingError(Codes.RightParenCode,
ParsingError.ActionOverItem.InsertAfter, s_tokens[s_tokens.Count -
1].endPosition + 1, message);
                                s_errors.Add(error);
                                throw new Exception(message);
                            }
                        }
                        else
                        {
                            Match(LexicalScanner.Codes.RightParenCode);
                        }
                        break;
                    case LexicalScanner.Codes.IntegerConstCode:
                        Match(LexicalScanner.Codes.IntegerConstCode);
                        break;
                    case LexicalScanner.Codes.DoubleConstCode:
                        Match(LexicalScanner.Codes.DoubleConstCode);
                        break;
                    case LexicalScanner.Codes.IdentifierCode:
                        Match(LexicalScanner.Codes.IdentifierCode);
                        break;
                    case LexicalScanner.Codes.LogicalConstantCode:
                        Match(Codes.LogicalConstantCode);
                        break;
                    default:
                        Match(LexicalScanner.Codes.IdentifierCode);
                        break;
                }
            }
            else
            {
                string message = "Некорректный токен";
                if (s_tokens.Count > 1)
                {
                    var error = new ParsingError(Codes.IdentifierCode,
ParsingError.ActionOverItem.InsertAfter, s_tokens[s_currentTokenIndex -
1].endPosition + 1, message);
                    s_errors.Add(error);
                }
                else
                {
                    var error = new ParsingError(Codes.ErrorCode,
ParsingError.ActionOverItem.Remove, s_tokens[s_currentTokenIndex -
1].endPosition, message);
                    s_errors.Add(error);
                    throw new Exception(message);
                }
            }
        }
    }
      </pre></p>
</body>
</html>
```