Програмний модуль містить такі змінні і константи для синтаксичного розбору:

Клас **SyntaxAnalyzerWithTable** - синтаксичний аналізатор

public Stack<int> stack - стек

private AutomatTable table - таблиця переходів

private void FIllTable() - функція заповнення таблиці переходів

public void ProcessLexemOnState(Lexem lexem, ref int lexemIterator, ref int stateNumber) - функція, що оброблює лексему на певному стані

public void AnalyzeLexems() - функція, що запускає синтаксичний аналіз

Клас **AutomatTable** - таблиця переходів

private Dictionary<int,State> states - список станів

public State StateWithNumber(int number) - функція отримання стану за його номером.

Клас **State** - стан State Machine

private List<Transition> transitions - список переходів за цим станом

public void Run(Lexem inputLexem, ref int StateIterator, ref int lexemsIterator) - функція виконання стану

Клас **Transition** - одиночний перехід

public bool RespondLexem(Lexem lexem) - повертає ТАК, якщо поточна лексема підходить для переходу

public void PerformLexem(Lexem lexem, ref int stateIterator, ref int lexemsIterator) - застосовує перехід

Програмний модуль містить такі змінні і константи для побудови ПОЛІЗу:

Клас **PolizAnalyzer** - клас будувача полізу

public void AnalyzeLexems() - будує поліз

public List<Lexem> Poliz - повертає побудований поліз

private void UseOperatorsBlock() - відкидує блок оголешння змінних та кінцеву лексему, залишаючи лише блок операторів

private void ProcessLexem() - обробляє лексему

private void ProcessOperatorLexem() - обробляє лексему-оператор

private void ProcessSeparator() - обробляє оператор "ENTER"

private void ProcessThen(Lexem lastStack) - обробляє оператор "then"

private void ProcessElse(Lexem lastStack) - обробляє оператор "else"

private void ProcessEndIf(Lexem lastStack) - обробляє оператор "endif"

private void ProcessComma() - обробляє оператор ","

private void ProcessFor() - обробляє оператор "for"

private void ProcessStep() - обробляє оператор "step"

private void ProcessTo() - обробляє оператор "to"

private void ProcessDo() - обробляє оператор "do"

private void ProcessNext() - обробляє оператор "next"

private void ProcessCloseScobe() - обробляє закриту дужку

private void ProcessEmptyStack() - обробляє ситуацію, коли стек порожній

private void ProcessOpenScobe() - обробляє відкриту дужку

private void ProcessLowPriorityLexem() - обробляє ситуацію, коли поточний оператор нижчий за пріорітетом, аніж той, що є в стеку

Клас **PolizOperarionsList** - зберігає пріорітети операцій

public bool isOperation(string operation) - повертає так, якщо лексема - операція

public int LexemPriority(Lexem lexem) - повертає пріорітет операції

Клас **PolizOperation** - пріорітет операції

public string Operation - операція

public int Priority - її пріорітет

Програмний модуль містить такі змінні і константи для виконання ПОЛІЗу:

клас **PolizCompiler** - компілятор полізу

private int LastInputValue - останнє введене значення в поле вводу

private void DidConfirmInputDialog (object obj, ResponseArgs args) - обробник події "число введене"

public void Compile() - виконує компіляцію

private int CalculateExpression(List<Lexem> poliz, int start, int end) - викраховує математичне значення у вказаному полізі, на вказаному діапазоні

private bool CalculateLogicalExpression(int start, int end) - викраховує логічне значення на вказаному діапазоні

клас **LabelFinder** - віднаходить потрібну мітку

public static int PositionOpenLabel(List<Lexem> poliz, string label) - віднаходить потрібну відкриту мітку у заданому полізі

public static int PositionCloseLabel(List<Lexem> poliz, string label) - віднаходить потрібну закриту мітку у заданому полізі.