Содержание

[Содержание 1](#_Toc101378548)

[1 Модуль ввода-вывода 3](#_Toc101378549)

[1.1 Описание модуля 3](#_Toc101378550)

[1.2 Проектирование 3](#_Toc101378551)

[1.3 Разработка 3](#_Toc101378552)

[1.4 Тестирование 5](#_Toc101378553)

[2 Лексический анализатор 6](#_Toc101378554)

[2.1 Описание модуля 6](#_Toc101378555)

[2.2 Проектирование 6](#_Toc101378556)

[2.2.1 CLexer 6](#_Toc101378557)

[2.2.2 CToken 6](#_Toc101378558)

[2.2.3 CConstToken 6](#_Toc101378559)

[2.2.4 CVariant 7](#_Toc101378560)

[2.2.5 CIntVariant, CRealVariant, CStringVariant, CBooleanVariant 7](#_Toc101378561)

[2.2.6 CIdentToken 7](#_Toc101378562)

[2.2.7 CKeyWordToken 7](#_Toc101378563)

[2.2.8 CEmptyToken 7](#_Toc101378564)

[2.3 Разработка 7](#_Toc101378565)

[2.3.1 CLexer 7](#_Toc101378566)

[2.3.2 CToken 7](#_Toc101378567)

[2.3.3 CConstToken 8](#_Toc101378568)

[2.3.4 CVariant 8](#_Toc101378569)

[2.3.5 CIntVariant, CRealVariant, CStringVariant, CBooleanVariant 8](#_Toc101378570)

[2.3.6 CIdentToken 8](#_Toc101378571)

[2.3.7 CKeyWordToken 8](#_Toc101378572)

[2.3.8 CEmptyToken 8](#_Toc101378573)

[2.4 Тестирование 8](#_Toc101378574)

[3 Синтаксический анализатор 10](#_Toc101378575)

[3.1 Описание модуля 10](#_Toc101378576)

[3.2 Проектирование 10](#_Toc101378577)

[3.3 Разработка 10](#_Toc101378578)

[3.4 Тестирование 10](#_Toc101378579)

[4 Нейтрализатор ошибок 11](#_Toc101378580)

[4.1 Описание 11](#_Toc101378581)

[4.2 Проектирование 11](#_Toc101378582)

[4.3 Разработка 11](#_Toc101378583)

[4.4 Тестирование 11](#_Toc101378584)

1. Модуль ввода-вывода

Представляет модуль ввода-вывода, позволяющий посимвольно считывать содержимое программы из файла или строки и накапливать ошибки компиляции, по окончанию обработки выводить их в файл или консоль.

* 1. Проектирование

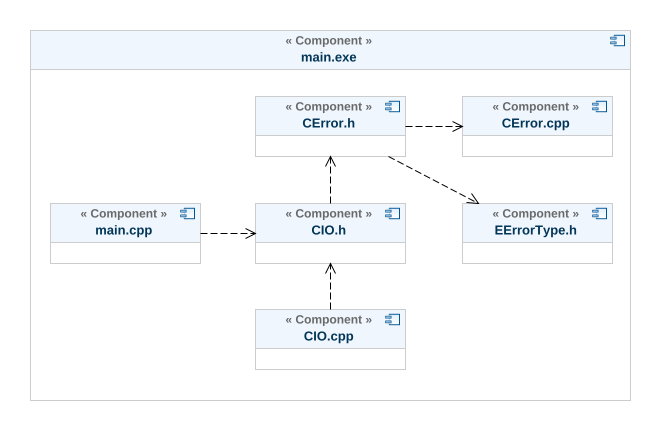


Диаграмма компонентов модуля ввода-вывода

Описание классов:

* CError – Класс ошибок, наследуемый от std::exception. Содержит поля номера строки и номера символа, а также перечисление типа ошибки.
* CIO – Основной класс модуля ввода-вывода, включает в себе поля указателей на потоки ввода и вывода, текущего номера строки, номера символа строки, буфера текущей обрабатываемой строки. Также включает в себя вектор указателей на объекты ошибок класса CError.

Перечисление EErrorType содержит возможные вариации ошибок.

* 1. Разработка

В качестве указателей используются unique\_ptr.

Для того, чтобы учесть различные способы хранения исходной программы, в качестве потока ввода использовался istream.

Методы класса CIO:

Открытые методы:

char GetNextChar() – получить следующий символ программы;

void AddError(EErrorType eType) – добавить новую ошибку в список ошибок, вызвать throw;

void PrintErrors() – вывести список ошибок; проверка на конец файла;

bool IsEndOfInput() – проверка окончания потока ввода;

string GetOutputString() – получить строку вывода (если для вывода был выбран вывод в строку, в других случаях возвращает пустую строку).

Закрытые методы:

void ReadString() – считать новую строку из файла в буфер.

Конструкторы класса модуля ввода-вывода:

CIO(ifstream& input) – ввод из файла, вывод в строку;

CIO(string input) – ввод из строки, вывод в строку;

CIO(ifstream &input, ostream &output) – ввод из файла, вывод ошибок в переданный поток;

CIO(string input, ostream &output) – ввод из строки, вывод ошибок в переданный поток.

Так как при вводе-выводе из файла или выводе в консоль не происходит выделения памяти, появляется необходимость ограничения её очищения, потому что в остальных случаях (чтение и вывод в строку) память выделяется. Для этого был создан деструктор, освобождающий указатели unique\_ptr, если ввод-вывод происходит из файла, во избежание освобождения памяти.

Метод получения следующего символа построчно считывает данные из входного потока, временно храня их в буфере, и возвращает следующий символ.

Метод добавления новой ошибки получает на вход тип ошибки и добавляет в вектор ошибок новую ошибку с переданным типом и с текущими в модуле ввода-вывода номерами символа и строки. Вызывает ошибку с помощью throw.

Метод вывода списка ошибок построчно выводит все ошибки из в поток вывода.

Метод чтения новой строки записывает в буфер новую строку, считанную с помощью функции getline(), также добавляет к считанной строке символ переноса.

Метод проверки на окончание файла возвращает значение false при условии, что поток достиг конца файла и номер символа буфера равен размеру буфера.

Метод получения строки вывода возвращает все данные, записанные в поток вывода при условии его использования.

Методы класса CError:

int GetLineNum() – получить номер строки ошибки;

int GetSymbolNum() – получить номер столбца ошибки;

string GetText() – получить текстовое описание ошибки.

Конструктор класса CError:

CError(int \_lineNumber, int \_symbolNumber, EErrorType \_eType).

* 1. Тестирование

Был создан класс IOUnitTest для unit тестов модуля ввода-вывода. Проверка на корректность осуществлялась с помощью функций Assert::AreEqual и Assert::IsTrue.

Методы тестирования:

IFileOString – проверка работы модуля ввода-вывода, где ввод происходил из файла, а вывод в строку;

IStringOString – проверка работы модуля ввода-вывода, где ввод и вывод происходил из строки;

IFileOFile – проверка работы модуля ввода-вывода, где ввод и вывод происходил из файла;

IStringOFile – проверка работы модуля ввода-вывода, где ввод происходил из строки, а вывод в файл;

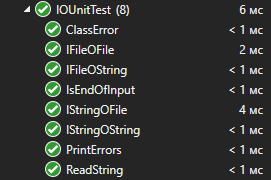
В каждом из перечисленных выше методов был считан один символ и выведена одна ошибка. Результаты сравнивались с прогнозируемыми.

ReadString – проверка считывания нескольких строк;

PrintErrors – проверка вывода нескольких ошибок;

IsEndOfInput – проверка работы метода IsEndOfInput класса CIO;

ClassError – проверка данных, хранящихся в классе CError.



Результат выполнения тестов класса IOUnitTest

1. Лексический анализатор

Представляет модуль лексического анализатора, который, используя модуль ввода-вывода, последовательно формирует набор токенов программы на языке программирования Паскаль. Также обрабатывает лексические ошибки языка.

* 1. Проектирование

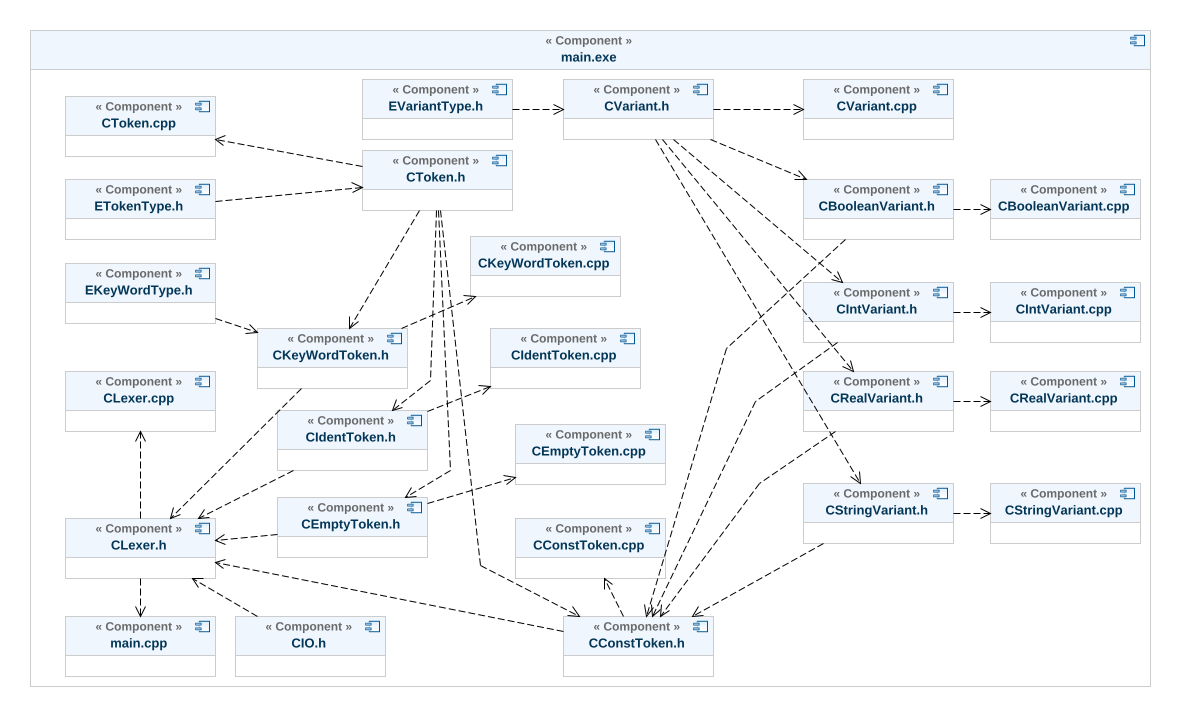


Диаграмма компонентов модуля лексичекого анализатора

* + 1. CLexer

Класс CLexer представляет основной класс модуля лексического анализатора, который позволяет получать токены из указанного модуля ввода-вывода.

Содержит поля указателя на модуль ввода-вывода, последнего считанного символа и словаря имен ключевых слов (ключами является элементы перечисления ETokenType).

* + 1. CToken

Абстрактный класс CToken представляет токен программы на языке Pascal. Содержит поле перечисления типа токена ETokenType (ttConst – константа, ttIdent – идентификатор, ttKeyword – ключевое слово).

* + 1. CConstToken

CConstToken – класс токена константы, является производным от CToken. Включает в себя указатель на объект класса CVariant, который характеризует тип и значение константы.

* + 1. CVariant

Абстрактный класс CVariant, представляющий константу. Содержит поле типа константы перечисления EVariantType (vtInt – целая, vtReal – вещественная, vtString – строковая, vtBoolean – булева).

* + 1. CIntVariant, CRealVariant, CStringVariant, CBooleanVariant

CIntVariant, CRealVariant, CStringVariant, CBooleanVariant – классы константы, являются производным от CVariant. Содержат поле значения константы.

* + 1. CIdentToken

CIdentToken – класс токена идентификатора, является производным от CToken. Включает в себя имя идентификатора.

* + 1. CKeyWordToken

CKeyWordToken – класс токена ключевого слова, является производным от CToken. Включает в себя поле ключевого слова перечисления EKeyWordType, которое содержит элементы для всех возможных ключевых слов языка Pascal в рамках реализуемой части компилятора.

* + 1. CEmptyToken

CEmptyToken – класс отсутствия токена, является производным от CToken.

Необходим в тех ситуациях, когда вернуть токен другого типа не удается: окончание потока ввода, при наличии в нем комментариев.

* 1. Разработка
     1. CLexer

Открытые методы:

CIO\* GetIOPtr() – получить следующий токен;

unique\_ptr<CToken> – получить модуль ввода-вывода.

Содержит единственный конструктор принимающий на вход указатель на модуль ввода-вывода.

Метод получения следующего токена считывает необходимую последовательность символов из потока ввода и на её основе строит объект одного из производных классов класса CToken – CConstToken, CIdentToken, CKeyWordToken, CEmptyToken.

Метод получения модуля ввода-вывода возвращает указатель на модуль ввода-вывода, содержащегося в лексическом анализаторе.

* + 1. CToken

Содержит конструктор принимающий тип токена.

Открытые методы:

ETokenType GetType() – получение типа токена;

virtual string ToString() = 0 – приведение к строке.

Также содержит виртуальный деструктор.

* + 1. CConstToken

Содержит переопределенный метод ToString() и метод CVariant\* GetVariant(), возвращающий указатель на CVariant;

* + 1. CVariant

Содержит конструктор принимающий тип константы.

Открытые методы:

EVariantType getVariantType() – получение типа константы;

virtual string ToString() = 0 – приведение к строке.

Также содержит виртуальный деструктор.

* + 1. CIntVariant, CRealVariant, CStringVariant, CBooleanVariant

Открытые методы:

GetValue() – получение значения;

string ToString() – переопределенный метод приведения к строке.

* + 1. CIdentToken

Содержит переопределенный метод ToString().

* + 1. CKeyWordToken

Открытые методы:

EKeyWordType GetKeyWordType () – получение типа ключевого слова;

string ToString() – переопределенный метод приведения к строке.

* + 1. CEmptyToken

Содержит переопределенный метод ToString().

* 1. Тестирование

Был создан класс LexerUnitTest для unit тестов лексического анализатора. Проверка на корректность осуществлялась с помощью функций Assert::AreEqual и Assert::IsTrue.

Методы тестирования:

OneSymbolKeyWordToken – проверка возвращаемых лексическим анализатором токенов односимвольных ключевых слов;

TwoSymbolsKeyWordToken – проверка возвращаемых лексическим анализатором токенов ключевых слов, которые распознаются после считывания дополнительного символа;

WordKeyWordToken – проверка возвращаемых лексическим анализатором токенов ключевых слов, которые являются цифробуквенными словами;

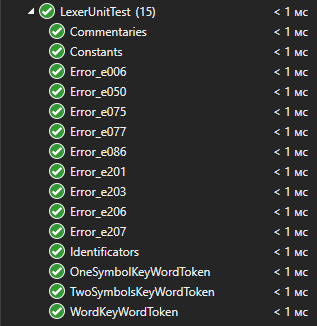
Commentaries – проверка пропуска комментариев;

Constants – проверка считывания констант различных типов;

Identificators – проверка считывания идентификаторов;

ErrorTestOutput – не тестируемый метод, возвращающий строку потока вывода на основании строки ввода, содержащую один токен с ошибкой;

Методы Error\_e006, Error\_e050, Error\_e075, Error\_e077, Error\_e086, Error\_e201, Error\_e203, Error\_e206, Error\_e207 проверяют содержимое потока вывода при получении соответствующей лексической ошибки.



Результат выполнения тестов класса LexerUnitTest

1. Синтаксический анализатор

Представляет модуль синтаксического анализатора, который, используя модули ввода-вывода и лексического анализатора, проверяет программу на языке программирования Паскаль на наличие синтаксических ошибок. Возвращает первую встретившуюся ошибку.

* 1. Проектирование

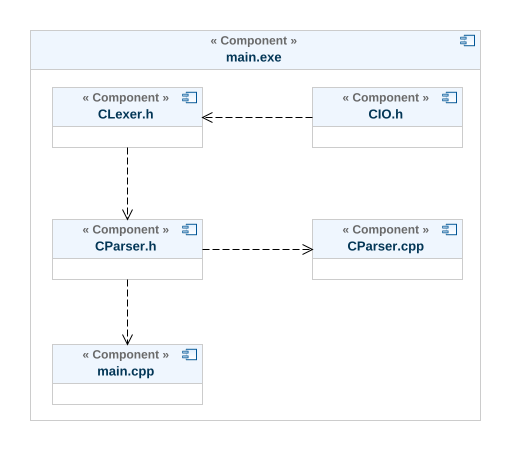


Диаграмма компонентов модуля синтаксического анализатора

CParser – Основной и единственный класс модуля синтаксического анализатора, включает в себе поля указателей на: объект класса CToken – последний, считанный лексическим анализатором токен; объект класса CIO – модуля ввода-вывода.

Содержит набор методов представляющих элементы БНФ, описывающей синтаксис языка Паскаль.

* 1. Разработка

Закрытые методы:

void GetNextToken() – записать следующий токен программы в поле curToken;

void Accept(ETokenType tType) – проверка последнего токена на принадлежность к переданному типу токена;

void Accept(EKeyWordType kwType) – проверка последнего токена ключевого слова на принадлежность к переданному типу;

bool IsRelOper() – является ли последний токен оператором отношения;

bool IsSign() – является ли последний токен оператором знака;

bool IsAddOper() – является ли последний токен оператором аддитивной операции;

bool IsMultOper() – является ли последний токен оператором мультипликативной операции;

bool EqualKeyWord(EKeyWordType kwType) – является ли последний токен токеном ключевого слова;

Открытые методы:

shared\_ptr<CLexer> GetLexerPtr() – получить указатель на объект лексического анализатора;

void Evaluate() – выполнить синтаксический анализ.

Содержит единственный конструктор принимающий shared указатель на объект лексического анализатора.

Методы Accept пробрасывают определенную ошибку в зависимости от переданного типа токена.

Методы БНФ-конструкций дублируют их, используя метод Accept для подтверждения наличия содержащихся в БНФ токенов и вызывая методы других БНФ-конструкций. Все эти методы являются закрытыми.

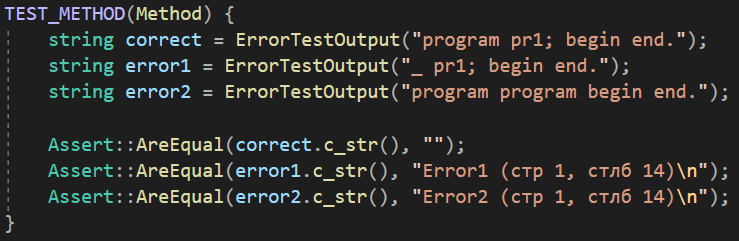
Содержит набор методов представляющих элементы БНФ, описывающей синтаксис языка Паскаль

* 1. Тестирование

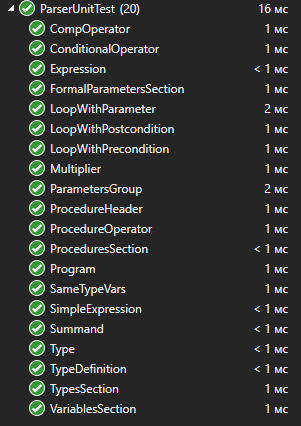
Был создан класс ParserUnitTest для unit тестов модуля синтаксического анализа. Проверка на корректность осуществлялась с помощью функции Assert::AreEqual.

Был создан метод ErrorTestOutput, принимающий на вход строку, которая является программой на языке Паскаль, и возвращающий строковое представление вывода ошибок.

Методы тестирования Program, TypesSection, TypeDefinition, Type, VariablesSection, SameTypeVars, ProceduresSection, ProcedureHeader, FormalParametersSection, ParametersGroup, CompOperator, Expression, SimpleExpression, Summand, Multiplier, ProcedureOperator, ConditionalOperator, LoopWithPrecondition, LoopWithPostcondition, LoopWithParameter проверяют корректность работы методов, описывающих соответствующие им БНФ-конструкции.



Формат теста методов синтаксического анализатора



Результат выполнения тестов класса ParserUnitTest

1. Нейтрализатор ошибок

На данном этапе синтаксический анализатор может находить только первую ошибку после чего анализ прекращается. Данное поведение не желательно, так как анализируемые программы могут содержать множество независимых друг от друга, обособленных ошибок.

* 1. Проектирование

После обнаружения ошибки необходимо пропустить определенное количество токенов, чтобы дойти до точки, с которой можно продолжить анализ. Для этого необходимо понимать, какие токены могут следовать за каждой БНФ-конструкцией. Данные токены назовем «внешними». Однако определение внешних токенов не зависит от БНФ-конструкций, а зависит от вызывающих их методов БНФ-конструкций.

Тогда удобнее всего будет передавать набор внешних токенов в метод БНФ-конструкции и после его выполнения в нем же провести проверку последнего считанного токена на принадлежность у переданному набору внешних токен. Если токен не принадлежит набору, в случае отсутствия ошибки в основной части метода выбросить ошибку о некорректном символе. После этого пропустить токены до первого встретившегося внешнего токена.

* 1. Разработка

bool Belong(vector<shared\_ptr<CToken>> tokens) – проверка принадлежности последнего токена переданному набору токенов;

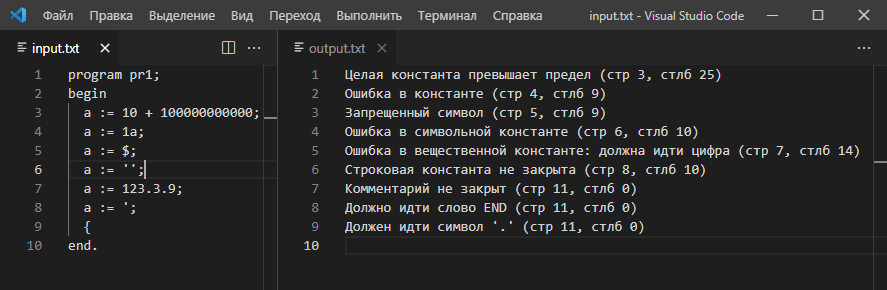
void SkipTo(vector<shared\_ptr<CToken>> followers) – считывание токенов до момента принадлежности последнего токена переданному набору токенов.

В класс CParser добавляется булево поле isNeutralizeErrors, означающее «отключение» нейтрализации ошибок.

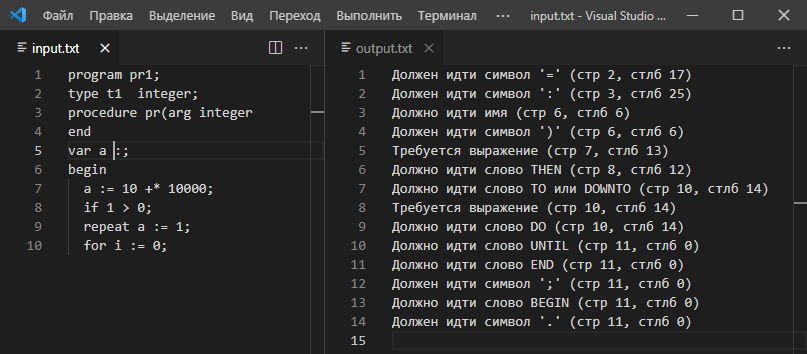
Все методы БНФ-конструкций модифицируются таким образом:

* В метод передается вектор внешних данной конструкции токенов;
* В каждый вызываемый в этом методе метод БНФ-конструкции передаются векторы внешних токенов (вектор внешних токенов дополняется при передаче в метод на основе вызываемых после этого метода методов БНФ-конструкции и Accept);
* Основная часть метода оборачивается в конструкцию try-catch;
* В каждый метод добавляется булева переменная catchedError инициализируемая значением false. В блоке catch переменной присваивается значение true;
* Добавляется проверка – если нейтрализация отключена и была найдена ошибка вызывается throw exception();
* Добавляется проверка – если нейтрализация включена и последний токен не содержится в векторе внешних токенов пропускаем все токены до первого принадлежащего внешним;
* Добавляется проверка – если нейтрализация включена и последний токен не содержится в векторе внешних токенов и не было найдено ошибки в основной части метода, вызываем метод AddError с кодом ошибки «Запрещенный символ». Так как идентифицировать, какой внешний символ должен быть следующим нет возможности.
  1. Тестирование

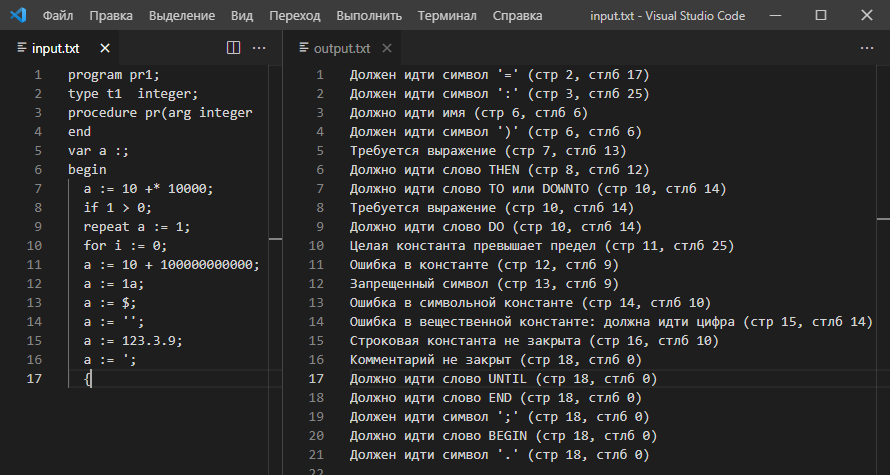
Были написаны три программы на языке Паскаль. Первая содержит все возможные лексические ошибки. Вторая содержит все возможные синтаксические ошибки (кроме отсутствия токена PROGRAM, которое прекращает синтаксический анализ в самом начале) и не содержит лексических ошибок. Третья объединяет первые две.



Обособленные лексические ошибки



Обособленные синтаксические ошибки



Большая часть лексических и семантических ошибок