Содержание

[Содержание 1](#_Toc101603795)

[1 Модуль ввода-вывода 3](#_Toc101603796)

[1.1 Проектирование 3](#_Toc101603797)

[1.2 Разработка 3](#_Toc101603798)

[1.3 Тестирование 5](#_Toc101603799)

[2 Лексический анализатор 6](#_Toc101603800)

[2.1 Проектирование 6](#_Toc101603801)

[2.1.1 CLexer 6](#_Toc101603802)

[2.1.2 CToken 6](#_Toc101603803)

[2.1.3 CConstToken 6](#_Toc101603804)

[2.1.4 CVariant 7](#_Toc101603805)

[2.1.5 CIntVariant, CRealVariant, CStringVariant, CBooleanVariant 7](#_Toc101603806)

[2.1.6 CIdentToken 7](#_Toc101603807)

[2.1.7 CKeyWordToken 7](#_Toc101603808)

[2.1.8 CEmptyToken 7](#_Toc101603809)

[2.2 Разработка 7](#_Toc101603810)

[2.2.1 CLexer 7](#_Toc101603811)

[2.2.2 CToken 7](#_Toc101603812)

[2.2.3 CConstToken 8](#_Toc101603813)

[2.2.4 CVariant 8](#_Toc101603814)

[2.2.5 CIntVariant, CRealVariant, CStringVariant, CBooleanVariant 8](#_Toc101603815)

[2.2.6 CIdentToken 8](#_Toc101603816)

[2.2.7 CKeyWordToken 8](#_Toc101603817)

[2.2.8 CEmptyToken 8](#_Toc101603818)

[2.3 Тестирование 8](#_Toc101603819)

[3 Синтаксический анализатор 10](#_Toc101603820)

[3.1 Проектирование 10](#_Toc101603821)

[3.2 Разработка 10](#_Toc101603822)

[3.3 Тестирование 11](#_Toc101603823)

[4 Нейтрализатор ошибок 13](#_Toc101603824)

[4.1 Проектирование 13](#_Toc101603825)

[4.2 Разработка 13](#_Toc101603826)

[4.3 Тестирование 14](#_Toc101603827)

[5 Семантический анализатор 16](#_Toc101603828)

[5.1 Проектирование 16](#_Toc101603829)

[5.1.1 CType 16](#_Toc101603830)

[5.1.2 CScope 17](#_Toc101603831)

[5.1.3 CScopes 17](#_Toc101603832)

[5.2 Разработка 17](#_Toc101603833)

[5.2.1 CType 17](#_Toc101603834)

[5.2.2 CStringType 17](#_Toc101603835)

[5.2.3 CRealType 17](#_Toc101603836)

[5.2.4 CProcedureType 17](#_Toc101603837)

[5.2.5 CNameType 17](#_Toc101603838)

[5.2.6 CIntType 17](#_Toc101603839)

[5.2.7 CBooleanType 17](#_Toc101603840)

[5.2.8 CCustomType 17](#_Toc101603841)

[5.2.9 CScope 17](#_Toc101603842)

[5.2.10 CScopes 17](#_Toc101603843)

[5.2.11 CParser 17](#_Toc101603844)

[5.3 Тестирование 17](#_Toc101603845)

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

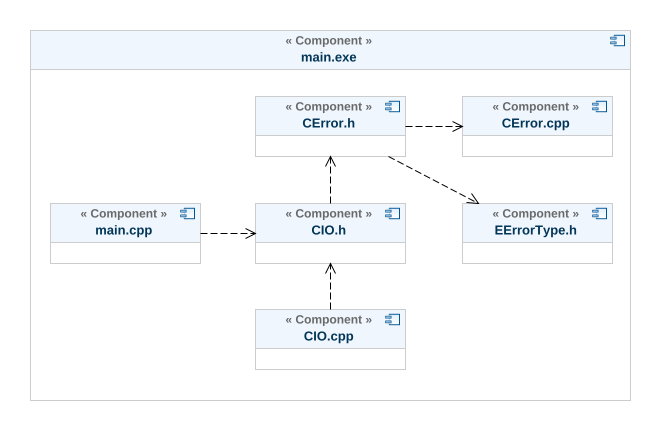


Диаграмма компонентов модуля ввода-вывода

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;

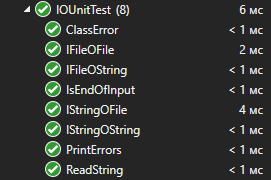
В каждом из перечисленных выше методов был считан один символ и выведена одна ошибка. Результаты сравнивались с прогнозируемыми.

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

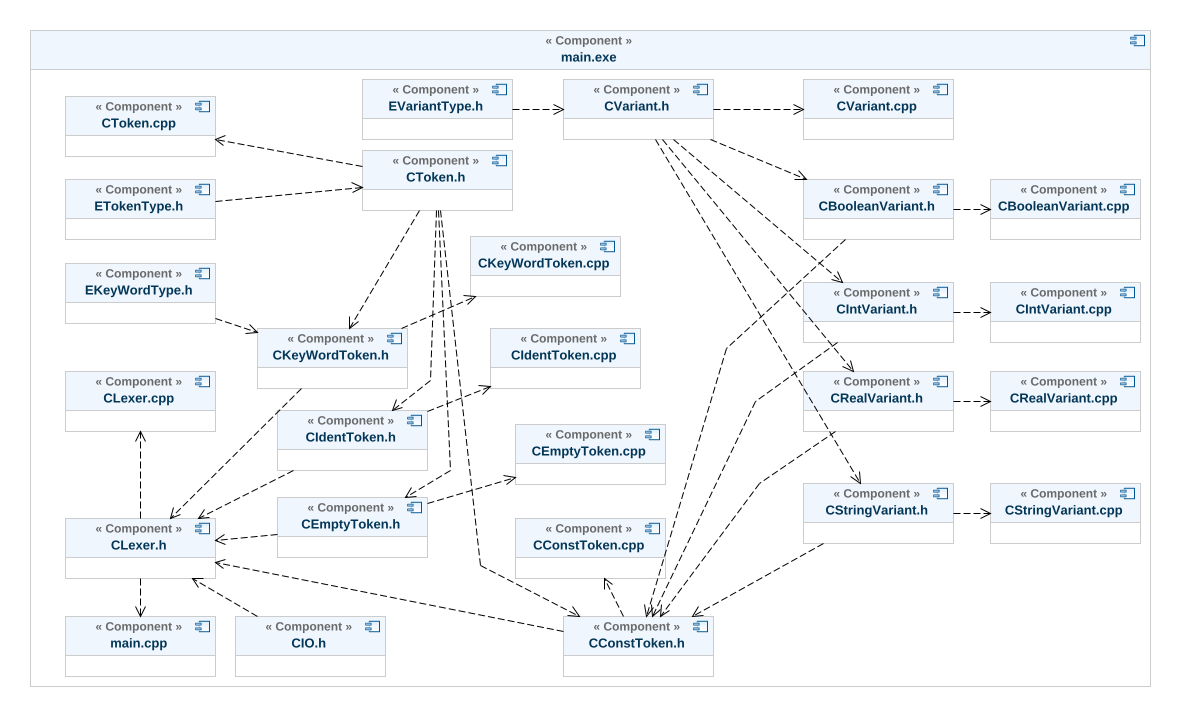


Диаграмма компонентов модуля лексичекого анализатора

* + 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;

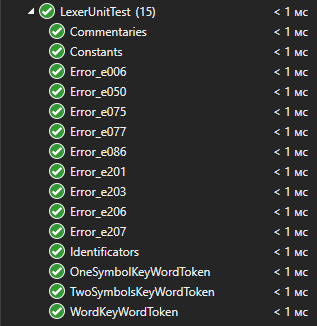
Commentaries – проверка пропуска комментариев;

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

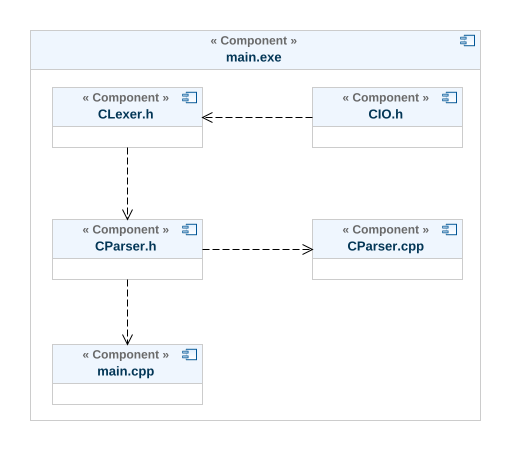


Диаграмма компонентов модуля синтаксического анализатора

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.

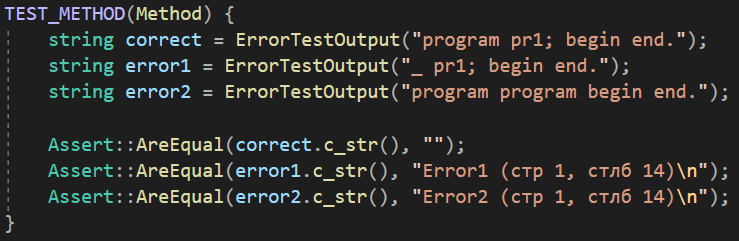
Содержит набор методов представляющих элементы БНФ, описывающей синтаксис языка Паскаль

* 1. Тестирование

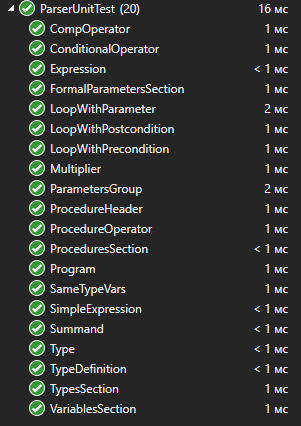
Был создан класс ParserUnitTest для unit тестов модуля синтаксического анализа. Проверка на корректность осуществлялась с помощью функции Assert::AreEqual.

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;

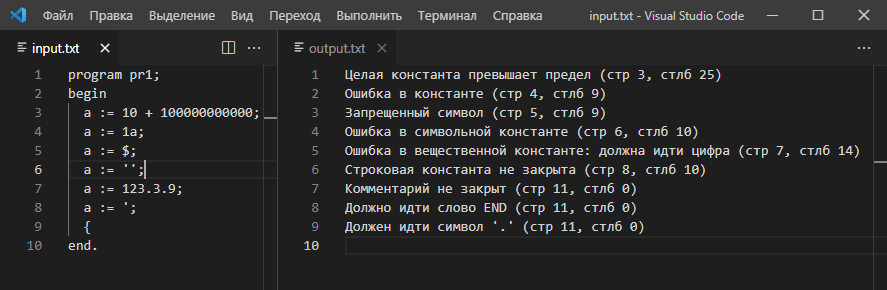
void SkipTo(vector<shared\_ptr<CToken>> followers) – считывание токенов до момента принадлежности последнего токена переданному набору токенов.

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

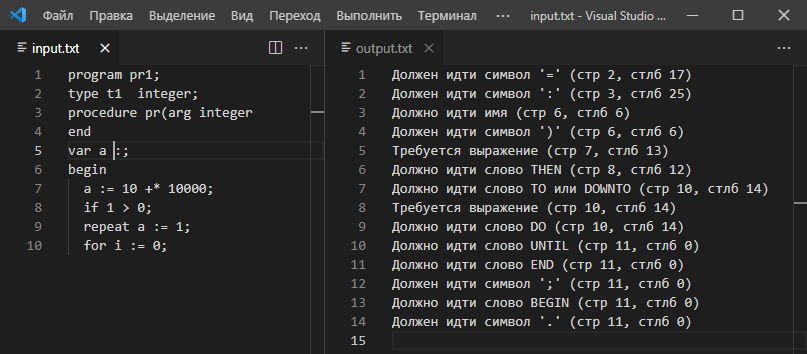
Все методы БНФ-конструкций модифицируются таким образом:

* 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

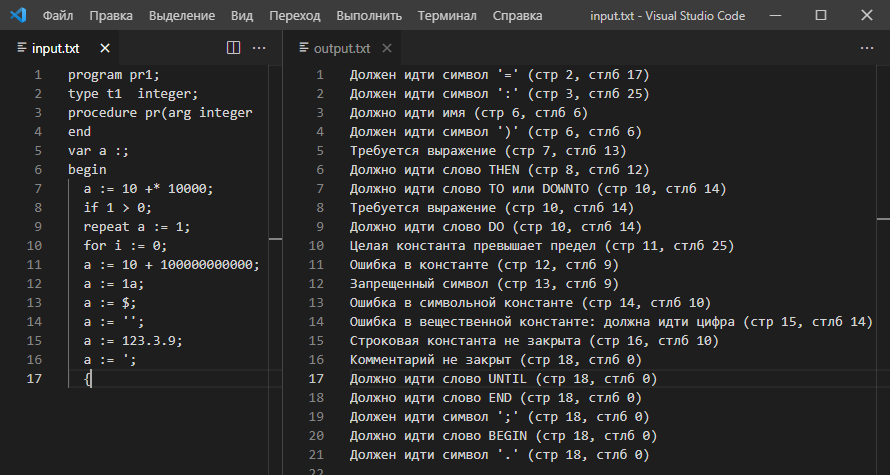
Были написаны три программы на языке Паскаль. Первая содержит все возможные лексические ошибки. Вторая содержит все возможные синтаксические ошибки (кроме отсутствия токена PROGRAM, которое прекращает синтаксический анализ в самом начале) и не содержит лексических ошибок. Третья объединяет первые две.



Обособленные лексические ошибки



Обособленные синтаксические ошибки



Большая часть лексических и семантических ошибок

1. Семантический анализатор

Является дополнением к синтаксическому анализатору, проверяя неформальные синтаксические правила, называемые контекстными условиями. Выделены следующие контекстные условия:

* В одной области видимости идентификатор может быть объявлен только один раз;
* Каждый нестандартный идентификатор (стандартные – integer, boolean, …) обязан быть объявлен перед его использованием;
* Контекстные условия соответствия типов величин, которые входят в синтаксические конструкции программ.
  1. Проектирование

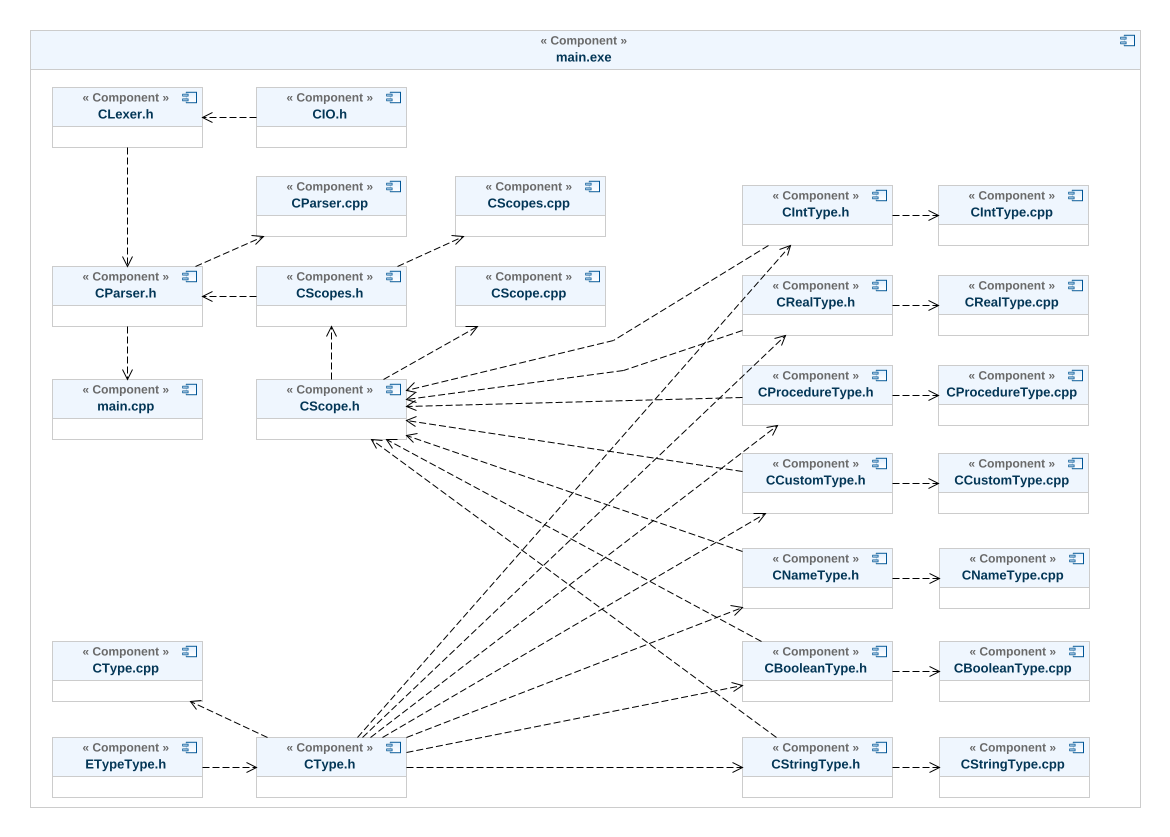


Диаграмма компонентов семантического анализатора

Так как семантический анализ неразрывно связан синтаксическим, то имеет место его интеграция в уже существующий модуль синтаксического анализатора CParser.

* + 1. CType

Абстрактный класс CType представляет тип идентификатора. Содержит поле перечисления ETypeType. Является родительским классом для классов:

* CStringType – класс типа строки;
* CRealType – класс типа вещественного числа;
* CProcedureType – класс типа процедуры. Содержит поле списка типов параметров процедуры;
* CNameType– класс типа идентификатора без назначения (имя программы);
* CIntType – класс типа вещественного числа;
* CBooleanType – класс логического типа;
* CCustomType – класс типа производного от другого типа. Содержит методы, возвращающие родительский тип и изначальный для цепочки типов родительский тип.
  + 1. CScope

Класс области видимости идентификаторов. Содержит ассоциативный массив всех идентификаторов данной области, а также хранит указатель на объект внешней области видимости.

* + 1. CScopes

Класс, являющийся стеком областей видимости CScope. Хранит указатель на текущую область видимости. Позволяет добавлять области в стек и извлекать их из стека.

* 1. Разработка
     1. CType

Содержит открытый метод возврата типа ETypeType GetType(), конструктор принимающий тип и виртуальный деструктор.

* + 1. CStringType, CRealType, CIntType, CBooleanType, CNameType

Содержат только конструкторы.

* + 1. CProcedureType

Открытые методы:

* void SetAttributes(vector< shared\_ptr<CType>> \_attributes) – метод передачи типов атрибутов в поле attributes;
* vector< shared\_ptr<CType>> GetAttributes() – метод получения вектора типов атрибутов.

Содержит конструктор, принимающий вектор типов атрибутов процедуры.

* + 1. CCustomType

Открытые методы:

* shared\_ptr<CType> GetSourceType() – метод, возвращающий значение поля sourceType (родительский тип);
* shared\_ptr<CType> GetOriginType() – метод, возвращающий изначальный родительский тип всей цепочки типов;
* bool IsType() – метод, возвращающий значение поля isType.

Содержит конструктор, принимающий список типов атрибутов и булеву переменную, означающую, является ли сам идентификатор, к которому указывается этот тип, типом.

* + 1. CScope

Открытые методы:

* shared\_ptr<CScope> GetScopeParent() – возвращает указатель на внешнюю область видимости;
* void SetScopeParent(shared\_ptr<CScope> \_parent) – принимает указатель на внешнюю область видимости;
* shared\_ptr<CType> AddIdent(sstring name, shared\_ptr<CType> ident) – добавляет новый идентификатор и его тип в ассоциативный контейнер;
* shared\_ptr<CType> LookupIdent(string name) – возвращает тип идентификатора по его имени. Если идентификатор не содержится в текущей области видимости, будет вызван этот же метод для внешней области видимости.

Содержит конструктор, принимающий указатель на внешнюю область видимости.

* + 1. CScopes

Открытые методы:

* shared\_ptr<CScope> GetCurrent() – возвращает текущую область видимости;
* shared\_ptr<CScope> Pop() – делает текущей область видимости внешнюю относительно текущей, возвращает бывшую область видимости;
* void Push(shared\_ptr<CScope> scope) – добавляет новую область видимости, делая её текущей, бывшую область записывает в поле parent новой области.

Содержит ничего не принимающий конструктор.

* + 1. CParser

Данный класс был дополнен новыми полями:

* unique\_ptr<CScopes> scopes – указатель на объект класса областей видимости;
* map<string, ETypeType> typeNames – ассоциативный контейнер, хранящий стандартные идентификаторы типов (integer, real, boolean, string) и соответствующие им объекты типов;

И новыми закрытыми методами:

* void AddIdent(string name, shared\_ptr<CType> type) – добавляет идентификатор с указанными именем и типом в текущую область видимости;
* shared\_ptr<CType> DeclaringCheckIdent() – проверяет наличие идентификатора, являющегося текущим токеном, в данной области видимости (включая внешние). Если находит, то возвращает его тип, если нет – вызывает ошибку «Имя не описано» и возвращает nullptr;
* string GetIdentName() – возвращает строковое представление текущего токена, если он является идентификатором;
* shared\_ptr<CType> GetTypeByName(string name) – возвращает тип по имени идентификатора, если данный идентификатор не является стандартным отсутствует в текущей области видимости, вызывает ошибку «Имя не описано» и возвращает nullptr;
* shared\_ptr<CType> GetConstantType() – возвращает тип константы на основе перечисления EVariantType;
* void AcceptProcedureCallTypes(shared\_ptr<CType> proc, vector<shared\_ptr<CType>> attributes) – проверяет количество переданных в процедуру при её вызове параметров и на соответствие их типов. Сравнивает атрибуты, содержащиеся в объекте proc класса CProcedureType, с переданными в метод;
* bool IsVariableType(shared\_ptr<CType> type) – является ли тип типом переменной (принадлежит одному из классов: CIntegerType, CRealType, CStringType, CBooleanType, CCustomType(если в истоке является одним из вышеуказанных));
* bool IsEqualTypes(shared\_ptr<CType> type1, shared\_ptr<CType> type2) – являются ли эти типы равными;
* bool IsScalarTypes(shared\_ptr<CType> type1, shared\_ptr<CType> type2) – являются ли эти типы скалярными (CIntegerType, CRealType);
* bool IsScalarType(shared\_ptr<CType> type) – является ли этот тип скалярным (CIntegerType, CRealType);
* bool IsBooleanTypes(shared\_ptr<CType> type1, shared\_ptr<CType> type2) – являются ли эти типы логическими;
* bool IsBooleanType(shared\_ptr<CType> type) – является ли этот тип логическим;
* bool IsIntegerTypes(shared\_ptr<CType> type1, shared\_ptr<CType> type2) – являются ли эти типы целыми;
* bool IsIntegerAndReal(shared\_ptr<CType> type1, shared\_ptr<CType> type2) – являются ли эти типы целым и вещественным;
* bool IsIntegerType(shared\_ptr<CType> type) – является ли этот тип целым;
* bool IsStringTypes(shared\_ptr<CType> type1, shared\_ptr<CType> type2) – является ли этот тип строковым;

Кроме этого, были модифицированы методы БНФ-конструкций для проверки соответствия контекстным условиям.

Методы, возвращающие тип(ы) идентификатор(а/ов):

* ProcedureHeader – возвращает вектор типов параметров процедуры при её объявлении;
* FormalParametersSection – возвращает вектор типов параметров процедуры при её объявлении;
* ParametersGroup – возвращает вектор типов параметров процедуры группы одного типа при её объявлении;
* AssignOperator – возвращает тип правой части оператора;
* Expression – возвращает тип выражения;
* SimpleExpression – возвращает тип выражения;
* Summand – возвращает тип выражения;
* Multiplier – возвращает тип выражения;
* ProcedureOperator – возвращает вектор типов параметров процедуры при её вызове;
* ActualParameter – возвращает тип параметра процедуры при её вызове.

Организация областей видимости:

* В начале метода Program вызывается метод Push объекта scopes, в конце метода вызывается метод Pop;
* В методе ProcedureDefinition метод Push вызывается перед началом объявления параметров процедуры, метод Pop вызывается после окончания работы метода Block.

Объявление идентификаторов. Вызов метода AddIdent для добавления новых идентификаторов:

* Имя программы в методе Program;
* Имя нового типа в методе TypeDefinition;
* Имя новой переменной в методе SameTypeVars;
* Имя новой процедуры в методе ProcedureDefinition;
* Имя аргумента процедуры в методе ParametersGroup.

Проверка факта объявления идентификатора. Вызов метода DeclaringCheckIdent:

* Идентификатор оператора присвоения и оператора вызова процедуры в методе SimpleOperator;
* Идентификатор в методе Multiplier;
* Идентификатор параметра цикла в методе LoopWithParameter.

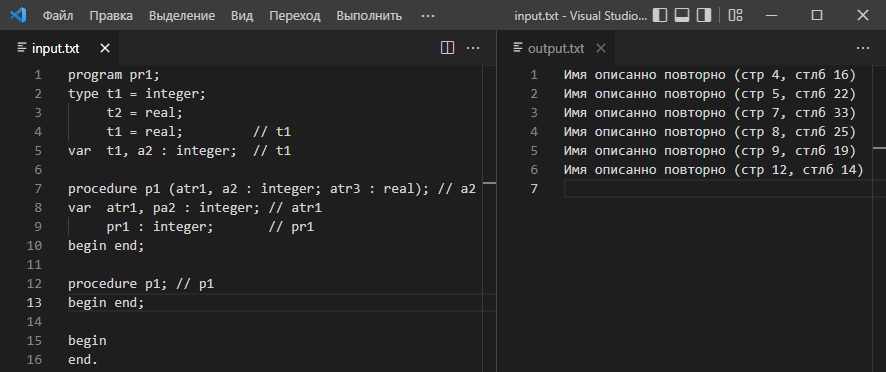
Проверка на соответствие типов с использованием методов IsVariableType, IsEqualTypes, IsScalarTypes, IsScalarType, IsBooleanTypes, IsBooleanType, IsIntegerTypes, IsIntegerType, IsStringTypes:

* Соответствие типов при присвоении в методе SimpleOperator;
* Соответствие типа идентификатора вызова процедуры типу процедуры в методе SimpleOperator;
* Соответствие типов при операциях сравнения в методе Expression;
* Соответствие типов при аддитивных операциях в методе SimpleExpression;
* Соответствие типов при мультипликативных операциях в методе Summand;
* Соответствие типа при операции not логическому в методе Multiplier;
* Соответствие типа выражения условия логическому в методах ConditionalOperator, LoopWithPrecondition, LoopWithPostcondition;
* Соответствие типа параметра цикла перечислимому (integer, boolean) в методе LoopWithParameter;
* Соответствие типов начального, конечного значения и параметра цикла в методе LoopWithParameter.

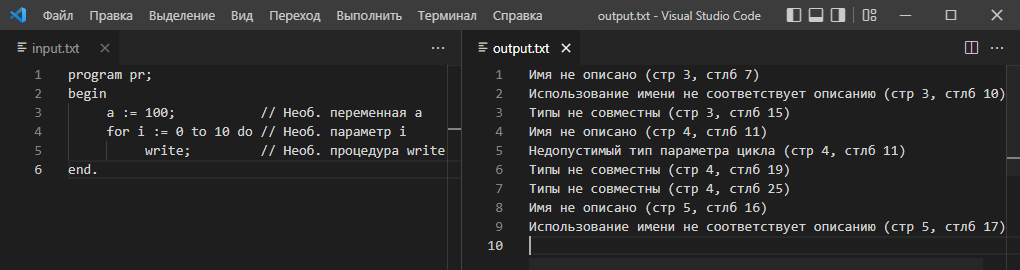
Проверка соответствия атрибутов процедуры при её вызове с помощью метода AcceptProcedureCallTypes в методе SimpleOperator.

* 1. Тестирование

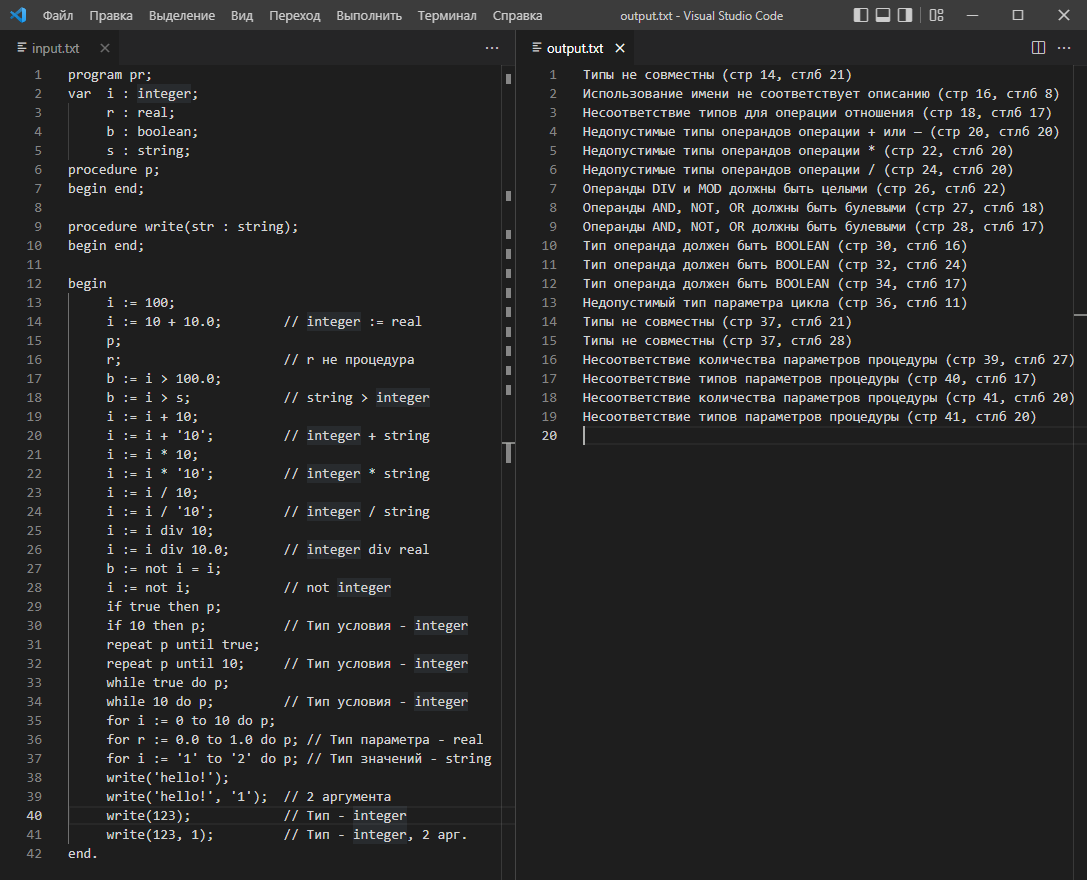
Были написаны 3 программы на языке Паскаль. Первая содержит различные варианты ошибок повторного описания идентификаторов. Вторая содержит различные варианта использования неописанных идентификаторов. Третья содержит различные варианты ошибок на несоответствие типов и ошибки вызовов процедур.



Различные варианты повторного описания идентификаторов



Различные варинты использования неописанных идентификаторов



Различные ситуации несоответствия типов и ошибки вызовов процедур