|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ | | | | | | | | | | | | |
|  | | | | | |  | | |  | | | |
| ФГБОУ ВО «Пермский государственный национальный исследовательский университет» | | | | | | | | | | | | |
|  | | | | |  | | | | |  | | |
|  | | ОТЧЕТ  По разработке компилятора для языка Pascal  Вариант 2 | | | | | | | | |  | |
|  | | | |  | | | | | |  | | |
|  | Работу выполнил  студент гр. ПМИ-3  Антонов А.А.  (подпись)  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2022 | | | | | |  | Проверил  Преподаватель кафедры МОВС  Пономарев Ф.А.  (подпись)  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2022 | | | |  |
|  | | | | | | | | | | | | |
|  | | | Пермь 2022 | | | | | | |  | | |

СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc97384054)

[1 Постановка задачи 3](#_Toc97384055)

[2 Структура компилятора 5](#_Toc97384056)

[3 Описание языка Pascal 8](#_Toc97384057)

[4 Модуль ввода-вывода 10](#_Toc97384058)

[4.1 Модуль ввода 10](#_Toc97384059)

[4.1.1 Проектирование 10](#_Toc97384060)

[4.1.2 Тестирование 10](#_Toc97384061)

[4.2 Модуль вывода 14](#_Toc97384062)

[5 Лексический анализатор 15](#_Toc97384063)

[5.1 Проектирование 15](#_Toc97384064)

[5.2 Токены 15](#_Toc97384065)

[6 Синтаксический анализатор 17](#_Toc97384066)

[7 Семантический анализатор 18](#_Toc97384067)

[8 Генерация кода 19](#_Toc97384068)

1 Постановка задачи

Глобальное задание: написать компилятор для подмножества языка Паскаль. Задание разбивается на отдельные этапы:

1. Модуль ввода-вывода (8 баллов, оценивается совместно с лексическим анализатором).
2. Лексический анализатор (12 баллов, оценивается совместно с модулем ввода-вывода).
3. Синтаксический анализатор (12 баллов) с нейтрализацией синтаксических ошибок (8 баллов).
4. Семантический анализатор с нейтрализацией семантических ошибок (20 баллов).
5. Генерация кода (25 баллов).

Для получения минимального проходного балла необходимо реализовать указанные этапы для подмножества языка Паскаль, описанного далее в разделе «*Общая минимальная часть*».

Для получения 70% баллов за анализаторы и 100% баллов за генерацию кода необходимо дополнительно реализовать конструкции, описанные далее в разделе «*Общая дополнительная часть*».

Для получения 100% баллов за анализаторы необходимо дополнительно реализовать анализ конструкций, описанных далее в разделе «*Индивидуальная часть*», в соответствии с заданным вариантом.

В случае не сдачи студентом задания по генерации кода на последнем лабораторном занятии (или ранее), данное задание для данного студента заменяется теоретическим экзаменом, который оценивается исходя из максимума в 25 баллов.

Замена задания по генерации на теоретический экзамен возможна также в случае, если студента не устраивают баллы, полученные за задание по генерации. В этом случае набранные баллы обнуляются, и студент сдает экзамен на тех же условиях, что и студенты, не сдававшие задание по генерации кода вообще.

*Общая минимальная часть.* Основные разделы программы: раздел описания переменных, раздел операторов. Переменные стандартных типов (Boolean, integer, real, char). Числовые константы. Арифметическое выражение (в выражении допустимы только константы, переменные, операции +, –, \*, / и скобки). Оператор присваивания и составной оператор.

*Общая дополнительная часть.* Раздел описания типов. Выражение (полностью, включая арифметические, логические операции, сравнения и т.д., но только над константами и простыми переменными (не индексированные, не поля записи, не указатели)). Условный оператор (if). Оператор цикла с предусловием (while).

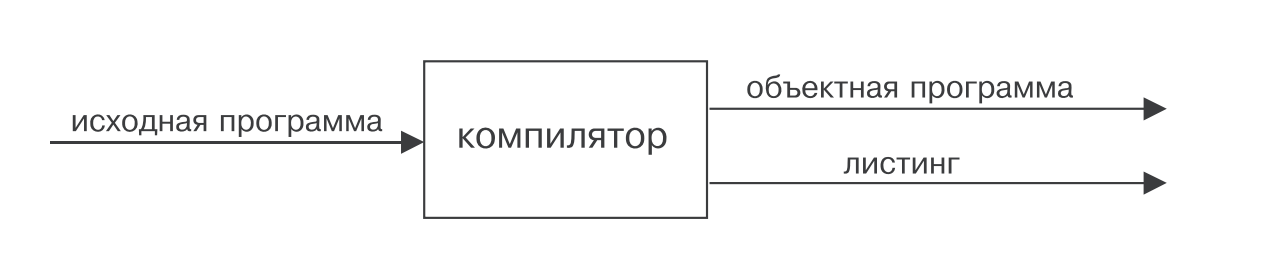
*Индивидуальная часть.*

Описание функций. Вызов функции в выражении. Оператор выбора (case).

2 Структура компилятора

Компилятор – это программа, которая переводит программу на языке высокого уровня в эквивалентную программу на другом (объектном) языке. Обычно также выдает листинг, содержащий текст исходной программы и сообщения обо всех обнаруженных ошибках.

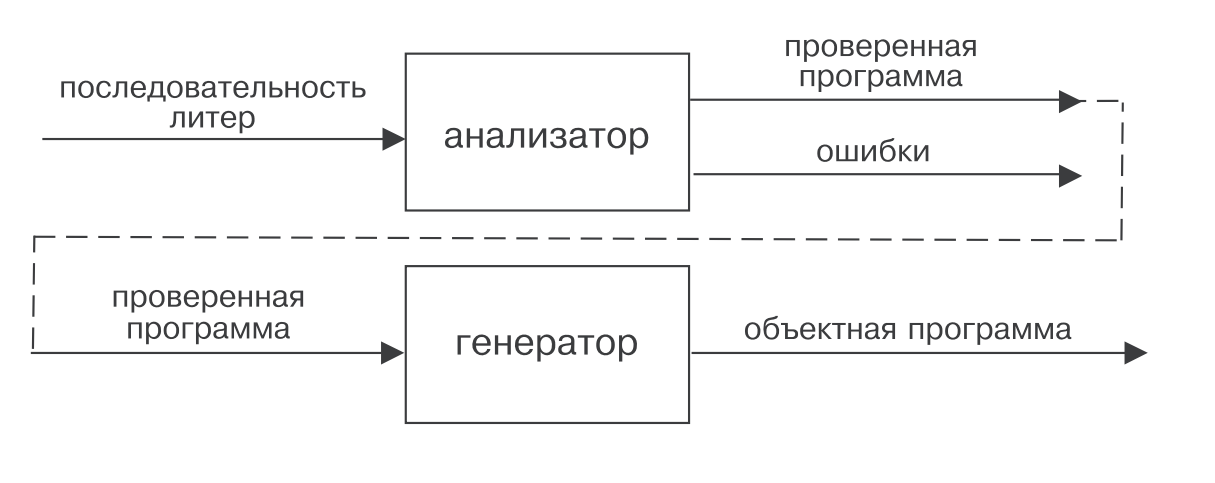
Компилятор можно представить в виде следующей схемы:



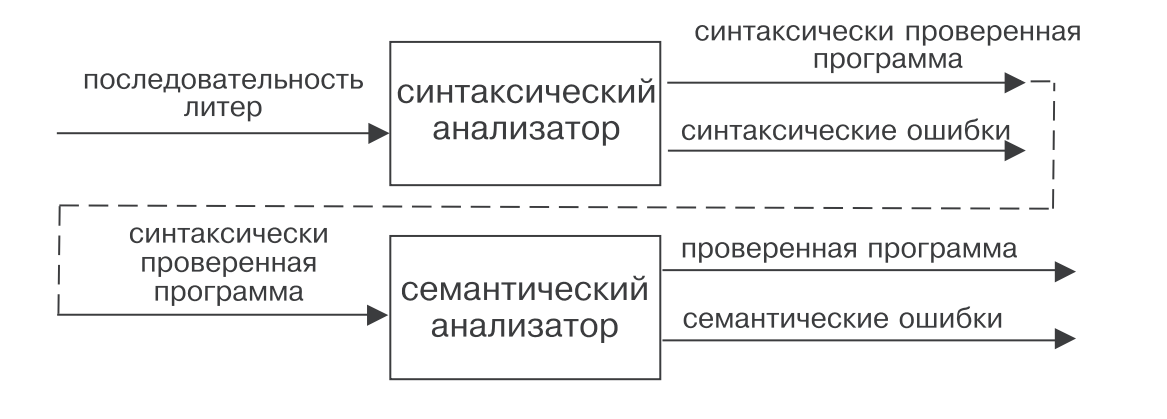
Работу компилятора можно разделить на 2 этапа:

1. Анализ – определение правильности исходной программы и формирование (в случае необходимости) сообщений об ошибках;
2. Синтез – генерация объектной программы; этот этап выполняется для программ, не содержащих ошибок.

Таким образом компилятор разбивается на следующие модули:



Анализатор в свою очередь можно разбиваться на синтаксический и семантический анализаторы:



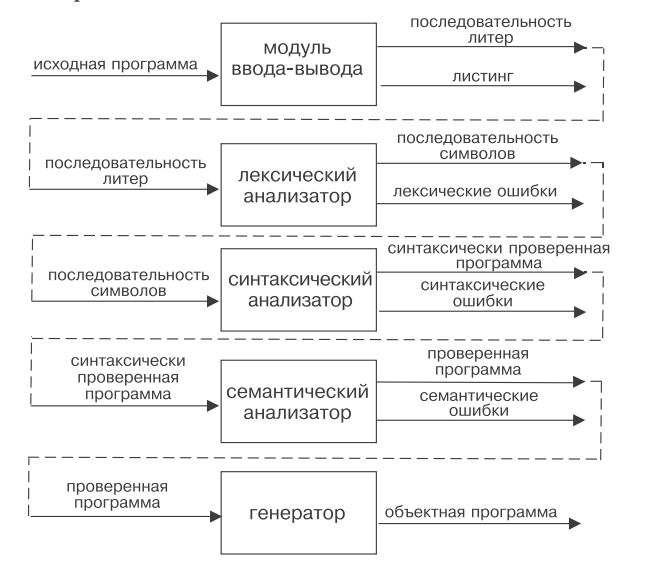
Синтаксический анализатор проверяет, удовлетворяет ли программа формальным правилам. Назначение же семантического анализатора состоит в том, чтобы выяснить, не нарушены ли неформальные правила описания языка.

Дальнейшее разбиение на модули выполняется внутри синтаксического анализатора.

Первый модуль – Лексический анализатор (Lexer) просматривает символы исходной программы и строит символы (лексемы) – идентификаторы, ключевые слова и константы.

Второй модуль – Синтаксический анализатор выполняет синтаксический анализ последовательности символов. На этом этапе символы рассматриваются как неделимые, и их представление как последовательности литер несущественно.

В результате получаем следующую структуру компилятора:



Для реализации компилятора необходимо реализовать каждый из модулей, представленных выше.

3 Описание языка Pascal

Будем рассматривать 3 типа токенов: ключевые слова, идентификаторы и константы.

Ключевые слова – это слова ЯП, которые имеют специальное, раз и навсегда закрепленное за ними значение. В программе нельзя использовать идентификаторы, совпадающие с ключевыми словами.

К ключевым словам будем также относить знаки операций, ограничители и специальные символы.

Идентификаторы – имена программ, модулей, функций, типов, переменных и констант. Идентификаторы начинаются с буквы (A-Z, a-z) или с символа нижнего подчеркивания и могут содержать буквы, символ нижнего подчеркивания и цифры. В языке Pascal идентификаторы нечувствительные к регистру.

Константы – это числа или строки, которые встречают в выражениях.

Ключевые слова:

|  |
| --- |
| program |
| begin |
| end |
| var |
| type |
| function |
| if |
| then |
| else |
| while |
| do |
| and |
| or |
| xor |
| not |
| + |
| - |
| \* |
| / |
| ( |
| ) |
| := |
| . |
| , |
| : |
| ; |
| = |
| < |
| > |
| <= |
| >= |
| <> |

4 Модуль ввода-вывода

4.1 Модуль ввода

4.1.1 Проектирование

Для реализации модуля ввода-вывода используется классы CInput и COutput, которые содержат следующие поля и методы:

class CInput {

private:

int currentLine;

int currentLinePosition;

std::ifstream inputFileStream;

public:

CInput(std::string inputFilePath);

std::shared\_ptr<CLiteral> NextChar();

};

class COutput {

private:

std::ofstream outputFileStream;

public:

COutput(std::string outFilePath);

void WriteErrorStd(std::shared\_ptr<CError> err);

void WriteErrorFile(std::shared\_ptr<CError> err);

};

inputFileStream и outputFileStream хранят ссылки на потоки файлового ввода и вывода из стандартной библиотеки соответственно.

В currentLine и currentLinePosition поддерживаются номер строки и позиция последнего считанного символа в этой строке соответственно.

В качестве аргументов конструктора классов принимаются строки с путем к исходному коду программы на языке Pascal и путем к файлу в который будут выводится ошибки в текстовом виде.

Метод NextChar возвращает указатель на объект класса CLiteral.

Класс CLiteral инкапсулирует в себе считываемые символы и их позицию.

class CLiteral {

public:

int lineNumber, linePosition;

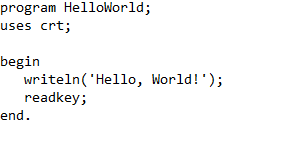
char c;

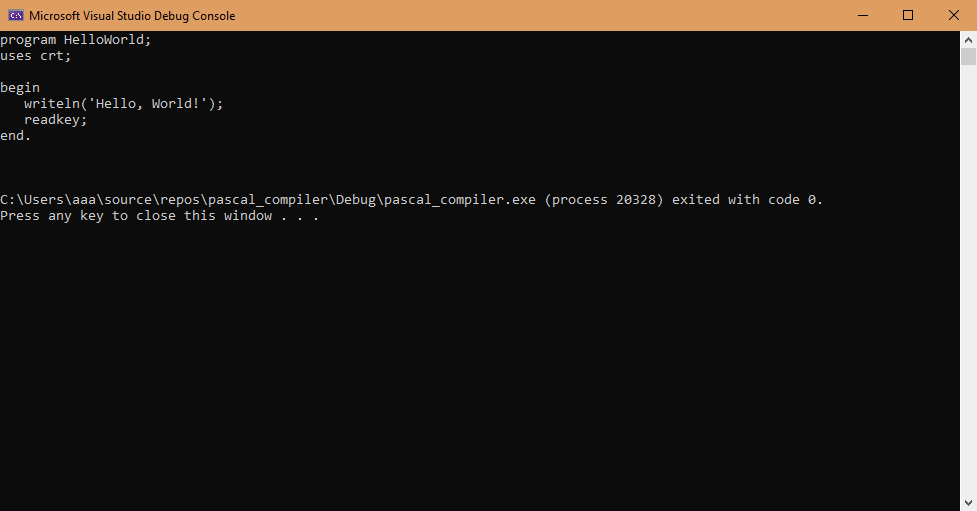
CLiteral(char \_c, int \_lineNumber, int \_linePosition);

};

4.1.2 Тестирование

Для тестирования рассмотрим пример паскаль программы и посимвольно выведем все, что возвращает нам модуль ввода.





Текст программы, который был выведен в консоль соответствует исходному.

Протестируем корректность сохранения позиции символа. Выведем каждый возвращаемый модулем символ вместе с его позицией в исходном тексте программы.







Позиции символов соответствуют позициям в исходном тексте программы на языке Pascal.

1. Лексический анализатор

Лексический анализатор формирует токены, а также удаляет комментарии. Лексический анализатор реализуется классом CLexer, который имеет следующее описание:

class CLexer

{

private:

std::unique\_ptr<CInput> cinput;

std::unique\_ptr<COutput> coutput;

bool isLetter(char c);

bool isDigit(char c);

bool isIgnored(char c);

char specialPrev;

std::shared\_ptr<CToken> readDigit();

std::shared\_ptr<CToken> readIdentifier();

std::shared\_ptr<CToken> readString();

std::shared\_ptr<CLiteral> curLiteral;

std::shared\_ptr<CToken> strToKeywordToken(std::string tokenStr);

std::shared\_ptr<CLiteral> GetNextLiteral();

public:

CLexer(CInput\* \_io);

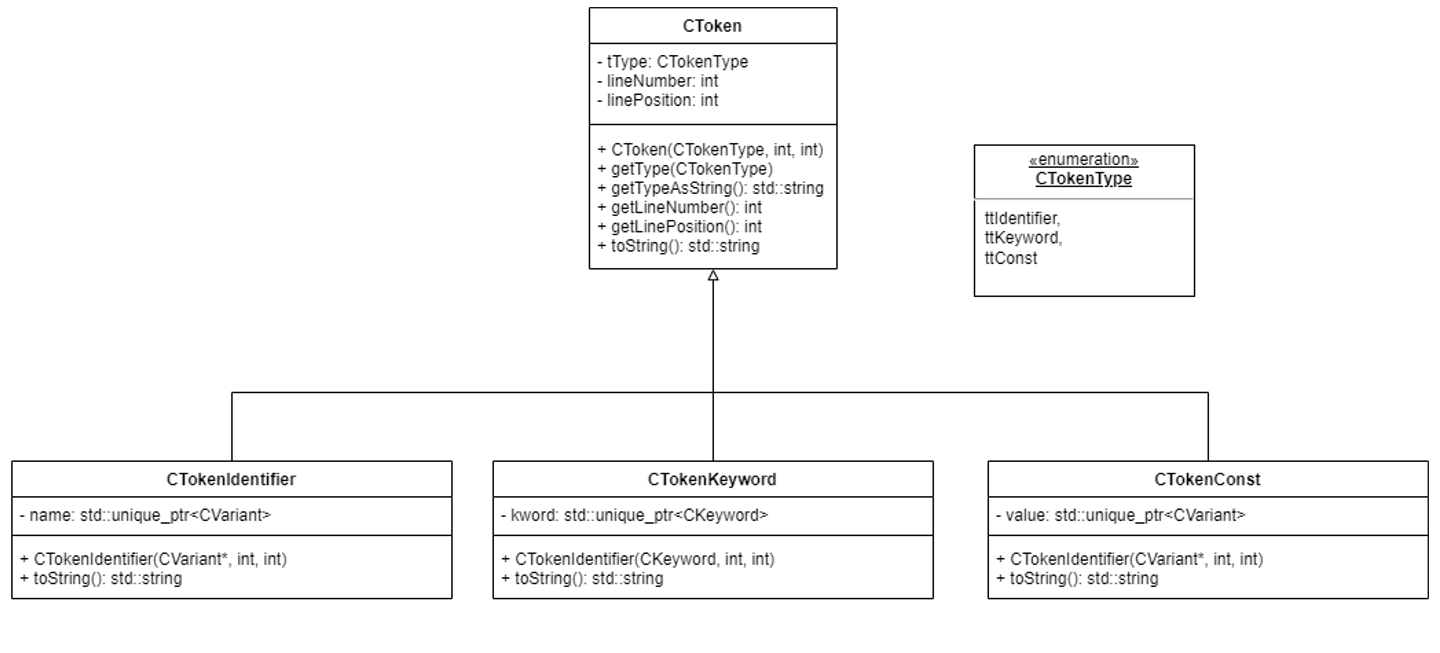
~CLexer();

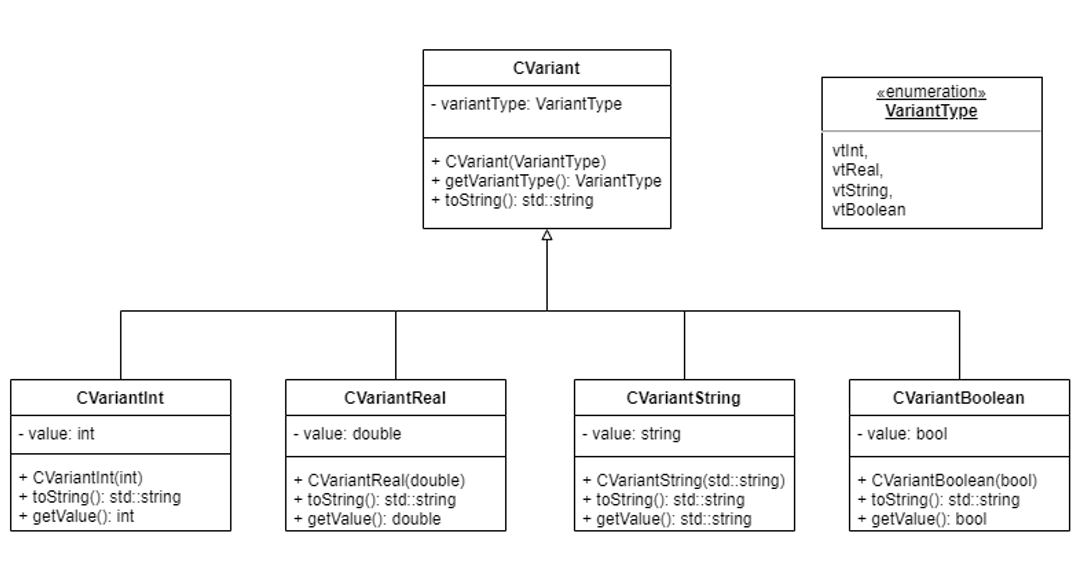
std::shared\_ptr<CToken> NextToken();

};

4.1 Токены

Ниже представлена диаграмма классов





* 1. Ошибки лексического анализатора

CErrorLexerUnknownLiteral – Неизвестный символ

CErrorLexerIncorrectNumber –Некорректная запись числа

CErorLexerUnmatchedCommentOpening – Отсутствует конец комментария

CErorLexerUnmatchedCommentEnding – Отсутствует начало комментария

CErrorLexerUnmatchedStringQuotes – Незавершенная запись строки

* 1. Тестирование

Для тестирования корректности разбития на токены, используем исходный код программы, который покрывает все случаи.

program gcd;

type

Bptr = boolean;

var a,b,c,t:integer;

inpt, outpt: string;

d: Bptr;

e: boolean;

f,h: real;

{

this is a comment

}

function GCD(m,n:integer):integer;

type

w = boolean;

var ee:integer;

function FFFF(m,n:integer):integer;

type

w1 = boolean;

var ee1:integer;

begin

while m<>n do

if m>n then m:=m-n else n:=n-m;

GCD:=m;

end;

begin

while m<>n do

if m>n then m:=m-n else n:=n-m;

GCD:=m;

end;

begin

inpt := 'Input 3 numbers';

e:= true;

d := e;

f := 123.4567;

a := (a+b)\*c - d/ ( GCD(64,18));

if (a = 30) and (b >= 15) or (c <= 17) xor (t <> 20) and not false then outpt := 'Fine';

case a = 30 of

true : outpt := 'Fine';

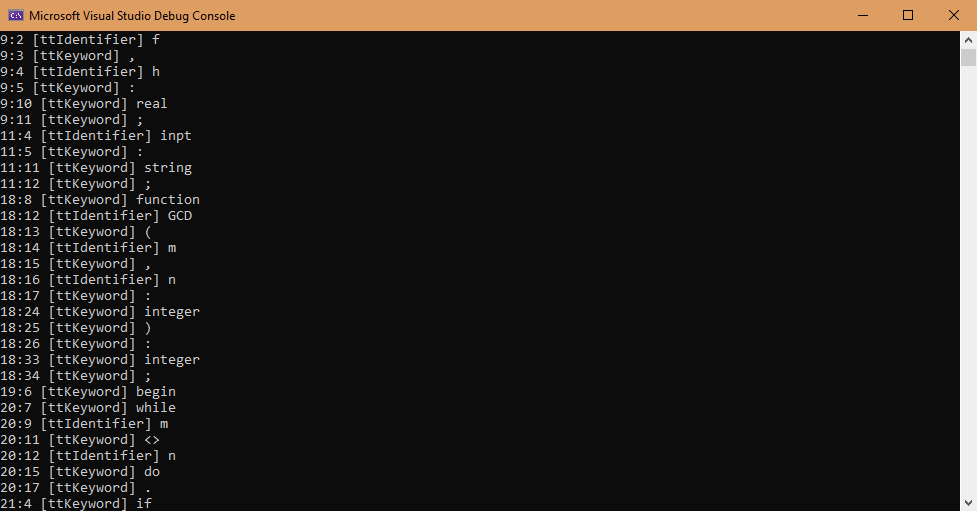
false: outpt := 'Fine';

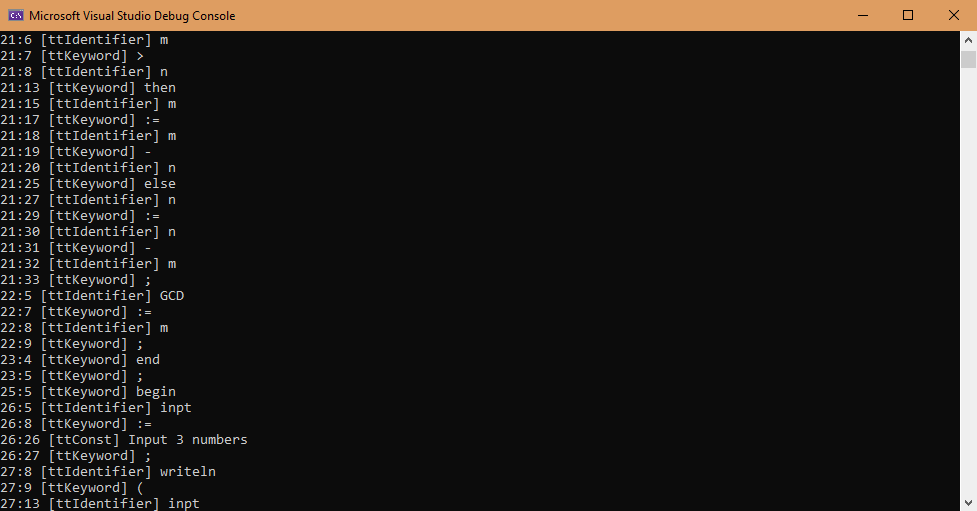
end;

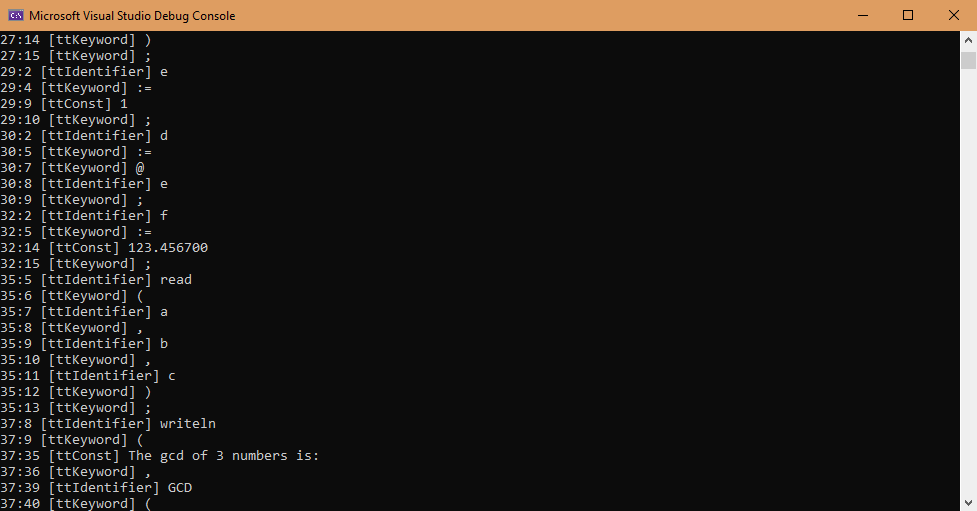
end.

Результат выполнения программы:













Места вызывающие ошибки выделены красным.

1. Отсутствует начало коментария

program gcd;

type

Bptr = boolean;

var a,b,c:integer;

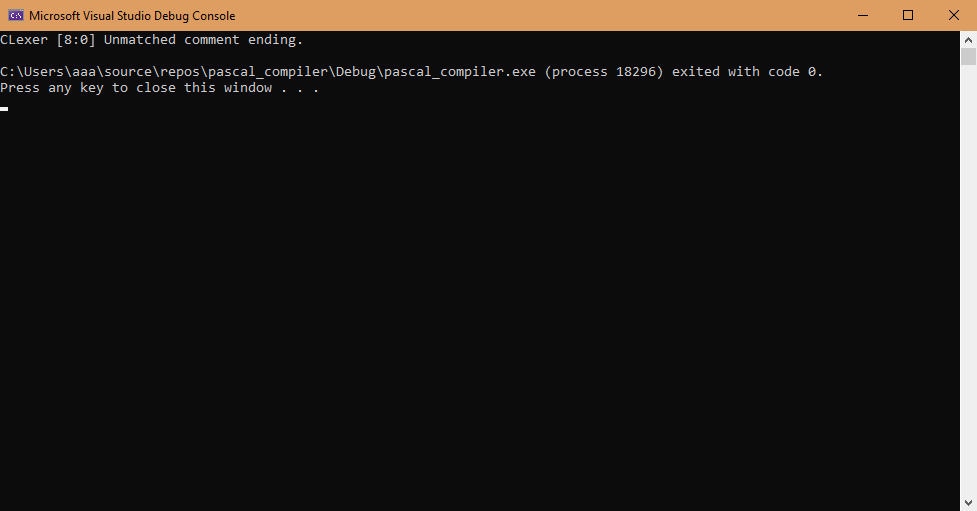
inpt:string;

}

begin

inpt := 'hi';

end.



1. Отсутствует конец комментария

program gcd;

{

type

Bptr = boolean;

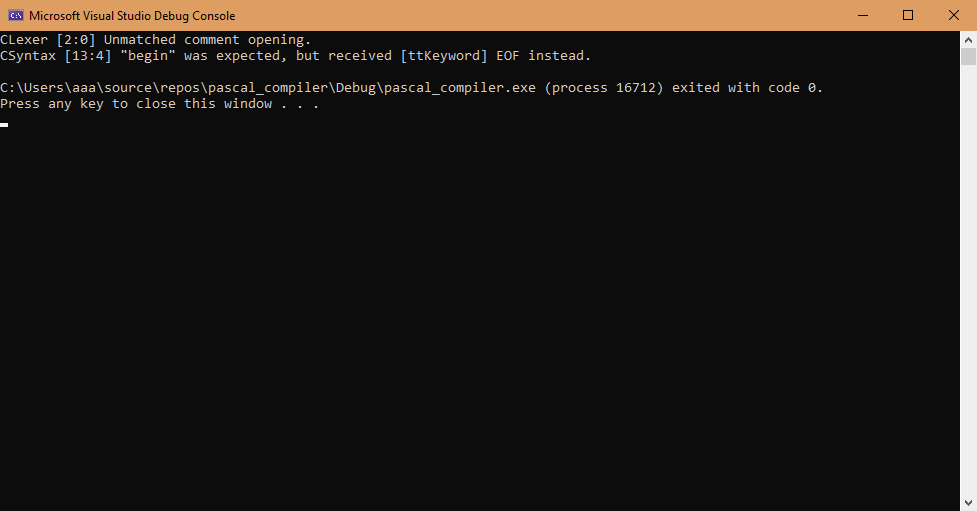
var a,b,c:integer;

inpt:string;

begin

inpt := 'hi';

end.



1. Неизвестный символ

program gcd;

type

Bptr = boolean;

var !a,b,c,t:integer;

inpt, outpt: string;

d: Bptr;

e: boolean;

f,h: real;

begin

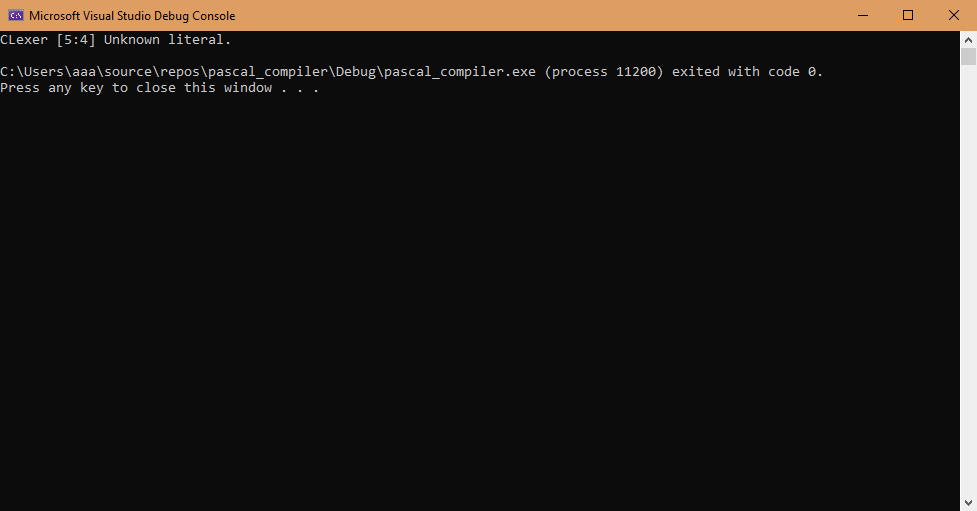
case a = 30 of

true : outpt := 'Fine';

false: outpt := 'Fine';

end;

end.



1. Неверное число

program gcd;

type

Bptr = boolean;

var !a,b,c,t:integer;

inpt, outpt: string;

d: Bptr;

e: boolean;

f,h: real;

begin

a := 123s213

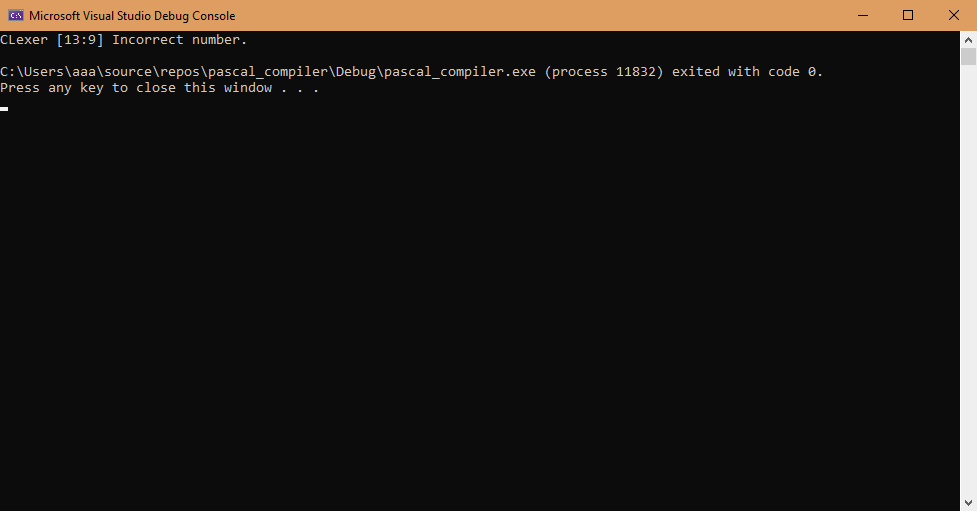
case a = 30 of

true : outpt := 'Fine';

false: outpt := 'Fine';

end;

end.



1. Отсутствует замыкающая кавычка в записи строки

program gcd;

type

Bptr = boolean;

var a,b,c,t:integer;

inpt, outpt: string;

d: Bptr;

e: boolean;

f,h: real;

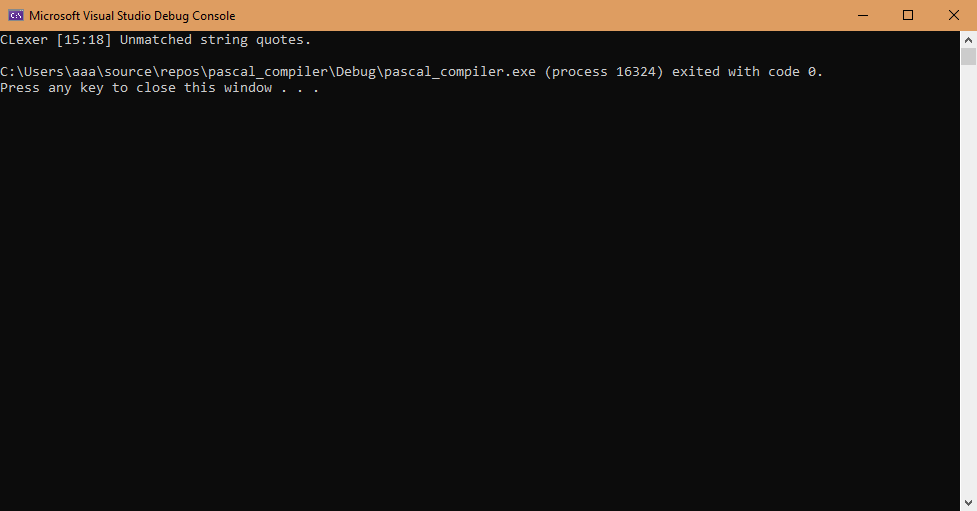
begin

case a = 30 of

true : outpt := 'Fine;

end;

end.



5 Синтаксический анализатор

5.1 Использованные БНФ

Ниже представлены использованные БНФ для языка Pascal.

<program> ::= program <identifier>;<block>.|<block>.

<block> ::= <typeDeclarationPart><varDeclarationPart><funcionDeclarationPart><statementPart>

<typeDeclarationPart> ::= <empty> | type <typeDeclaration>; {<typeDeclaration>;}

<typeDeclaration> ::= <identifier> = <type>

<type> ::= <identifier>

<varDeclarationPart>::= <empty> | var <varDeclaration>; {<varDeclaration>;}

<varDeclaration> ::= <identifier> {,<identifier>} : <type>

<funcionDeclarationPart> ::= {<funcionDeclaration>;}

<funcionDeclaration> ::= <functionHeading> <block>

<functionHeading> ::= function <identifier>:<type>; |

function <identifier>(<formalParameterSection>{;<formalParameterSection>}):<type>;

<formalParameterSection> ::= <empty> | <parameterGroup> | var <parameterGroup>

<parameterGroup> ::= <identifier>{,<identifier>}: <type>

<statementPart> ::= begin <compoundStatement> end

<compoundStatement> ::= <empty> | <statement>{;<statement>}

<statement> ::= <simpleStatement>|<structuredStatement>

<simpleStatement> ::= <assignmentStatement>

<assignmentStatement> ::= <variable> := <expression>

<variable> ::= <identifier>

<expression> ::= <simpleExpression>{<relationalOperator><simpleExpression>}

<relationalOperator> ::= <|>|>=|<=|=|<>

<simpleExpression> ::= <term>{<addingOperator><term>}

<addingOperator> ::= + | - | or | xor

<term> ::= <factor>{<multiplyingOperator><factor>}

<multiplyingOperator> ::= \* | / | and

<factor> ::= <variable> | <unsignedConst> | (<expression>) | <functionDesignator> | <unaryOperator><factor>

<unaryOperator> ::= + | - | not | @

<unsignedConst> ::= <unsignedNumber> | <stringConst>

<unsignedNumber> ::= <unsignedInteger> | <unsignedReal>

<functionDesignator> ::= <identifier>() | <identifier>(<actualParameter>{,<actualParameter>})

<actualParameter> ::= <expression>|<empty>

<empty> ::=

<structuredStatement> ::= <ifStatement>|<whileStatement>|<caseStatement>

<ifStatement> ::= if <expression> then <statement> | if<expression> then <statement> else <statement>

<whileStatement> ::= while <expression> do <statement>

<caseStatement> ::= case <expression> of <case list element> {; <case list element> } end

<caseListElement> ::= <empty> | <unsignedConst>{, <unsignedConst> } : <statement>

5.2 Общее описание

Синтаксический анализатор представлен в виде класса CSyntax.

Объекты данного класса содержат два поля - указатель на объект класса лексического анализатора и указатель на текущий рассматриваемый токен.

Для каждой БНФ в классе реализован метод, который ей соответствует по логике, т.е. вызывает методы других БНФ или обрабатывает терминальные символы. Реализация каждого метода сделана, учитывая следующее условие: если мы находимся в начале нового правила, то текущий токен - всегда самый первый токен в левой части БНФ.

Помимо методов, которые непосредственно реализуют логику синтаксического анализ, CSyntax содержит вспомогательные методы, название которых начинается с Accept или TryAccept. Данные методы обрабатывают индивидуальные токены и переходят к следующему. Методы начинающиеся с Accept выдают синтаксическую ошибку в случае неверного токена, методы начинающиеся с TryAccept в такой ситуации не переходят к следующему токену и возвращают false.

5.3 Нейтрализация синтаксических ошибок

В случае возникновения синтаксической ошибки она обрабатывается с помощью конструкции try catch. При обработке все последующие токены будут пропущены пока не встретиться токен, который позволит далее корректно проводить синтаксический анализ.

Пропуск токенов производиться с помощью метода класса CSyntax - SkipTo, на вход которому подается множество токенов, которые вызовут его остановку.

Пример работы:

program gcd;

type

Bptr = boolean;

var a,b,c, t:integer;

d: Bptr;

e: boolean;

inpt:string;

{

this is a comment

}

begin

if (a = 30) and (b >= 15) or (c <= 17) xor (t <> 20) and not {12} then outpt := 'Fine';

case a = {30} of

1 : outpt := 'Fine';

2: outpt := 'Fine';

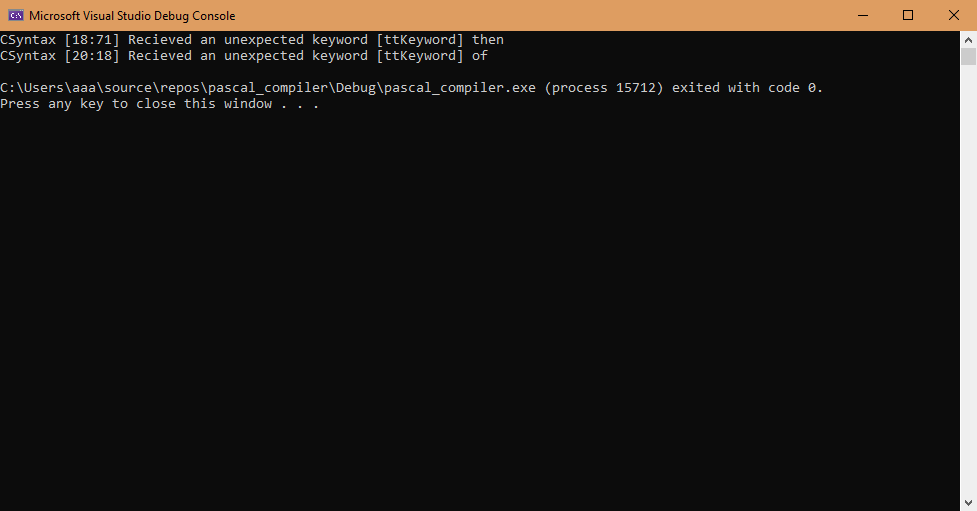
3: outpt := 'Fine';

end;

end.

Места, вызывающие ошибку, выделены красным.

Вывод:



5.4 Синтаксические ошибки

CErrorSyntaxUnexpectedKeyword – Встречено ключевое слово, когда ожидалось другое ключевое слово, константа или идентификатор.

CErrorSyntaxExpectedIdentifier – Ожидался идентификатор

CErrorSyntaxExpectedKeyword – Ожидалось некоторое ключевое слово

CErrorSyntaxExpectedConst – Ожидалась константа

5.5 Тестирование

Ошибки в исходном коде выделены красным.

1. Корректная программа

program gcd;

type

Bptr = boolean;

var a,b,c,t:integer;

inpt, outpt: string;

d: Bptr;

e: boolean;

f,h: real;

{

this is a comment

}

function GCD(m,n:integer):integer;

type

w = boolean;

var ee:integer;

function FFFF(m,n:integer):integer;

type

w1 = boolean;

var ee1:integer;

begin

while m<>n do

if m>n then m:=m-n else n:=n-m;

GCD:=m;

end;

begin

while m<>n do

if m>n then m:=m-n else n:=n-m;

GCD:=m;

end;

begin

inpt := 'Input 3 numbers';

e:= true;

d := e;

f := 123.4567;

a := (a+b)\*c - d/ ( GCD(64,18));

if (a = 30) and (b >= 15) or (c <= 17) xor (t <> 20) and not false then outpt := 'Fine';

case a = 30 of

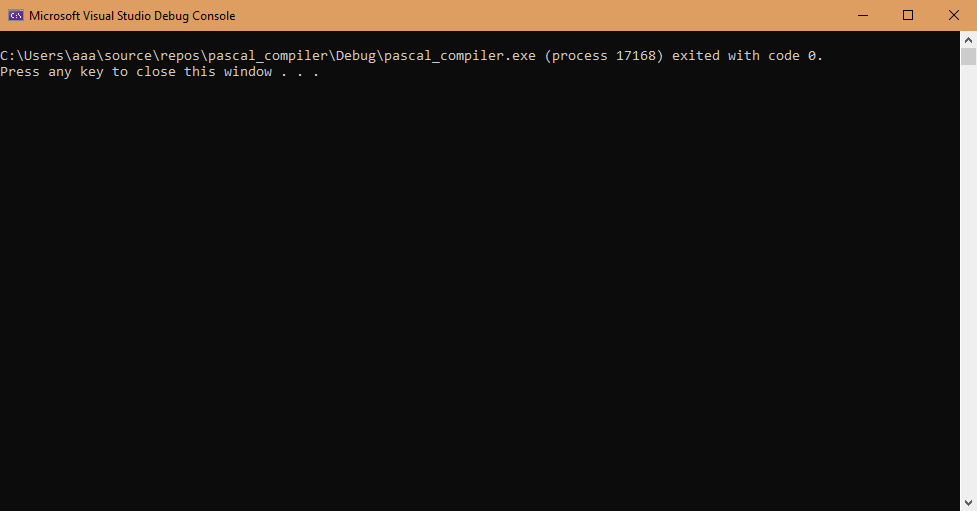
true : outpt := 'Fine';

false: outpt := 'Fine';

end;

end.

Вывод:



2. Пропущены некоторые операторы присваивания

program gcd;

type

Bptr = boolean;

var a,b,c,t:integer;

inpt, outpt: string;

d: Bptr;

e: boolean;

f,h: real;

{

this is a comment

}

function GCD(m,n:integer):integer;

type

w = boolean;

var ee:integer;

function FFFF(m,n:integer):integer;

type

w1 = boolean;

var ee1:integer;

begin

while m<>n do

if m>n then m:=m-n else n:=n-m;

GCD:=m;

end;

begin

while m<>n do

if m>n then m:=m-n else n:=n-m;

GCD:=m;

end;

begin

inpt := 'Input 3 numbers';

e:= true;

d {:=} e;

f {:=} 123.4567;

a {:=} (a+b)\*c - d/ ( GCD(64,18));

if (a = 30) and (b >= 15) or (c <= 17) xor (t <> 20) and not false then outpt := 'Fine';

case a = 30 of

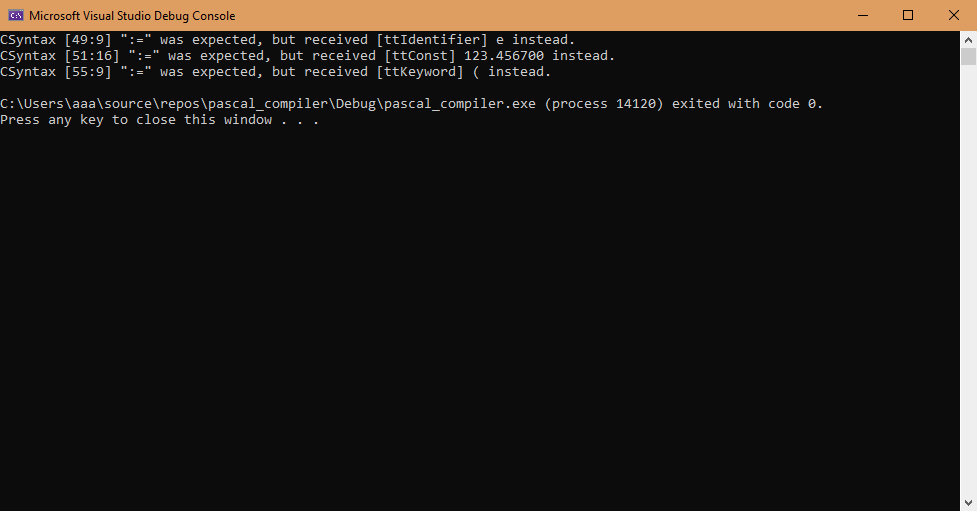
true : outpt := 'Fine';

false: outpt := 'Fine';

end;

end.

Вывод:



1. Пропущены идентификаторы в первых объявлениях типов и переменных.

program gcd;

type

{Bptr} = boolean;

var {f,h}: real;

a,b,c,t:integer;

inpt, outpt: string;

d: Bptr;

e: boolean;

{

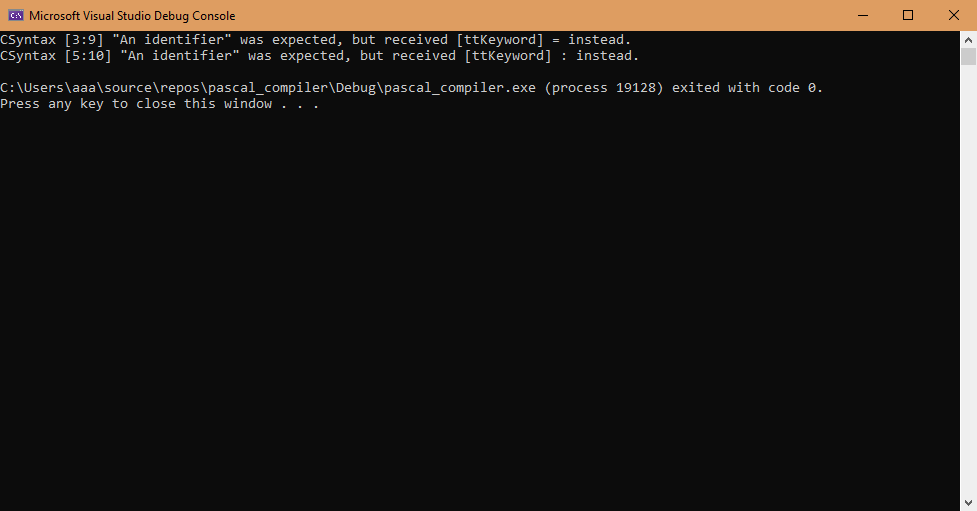
this is a comment

}

begin

end.

Вывод:



1. Пропущено ключевое слово of

program gcd;

type

Bptr = boolean;

var a,b,c,t:integer;

inpt, outpt: string;

d: Bptr;

e: boolean;

f,h: real;

begin

case a = 30 {of}

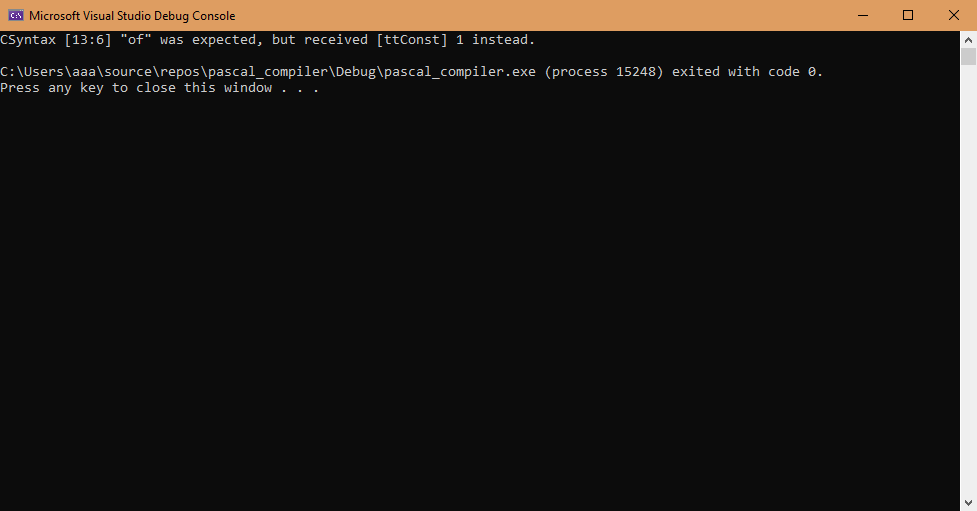
true : outpt := 'Fine';

false: outpt := 'Fine';

end;

end.

Вывод:



1. Пропущена константа в конструкции case

program gcd;

type

Bptr = boolean;

var a,b,c,t:integer;

inpt, outpt: string;

d: Bptr;

e: boolean;

f,h: real;

begin

case a = 30 of

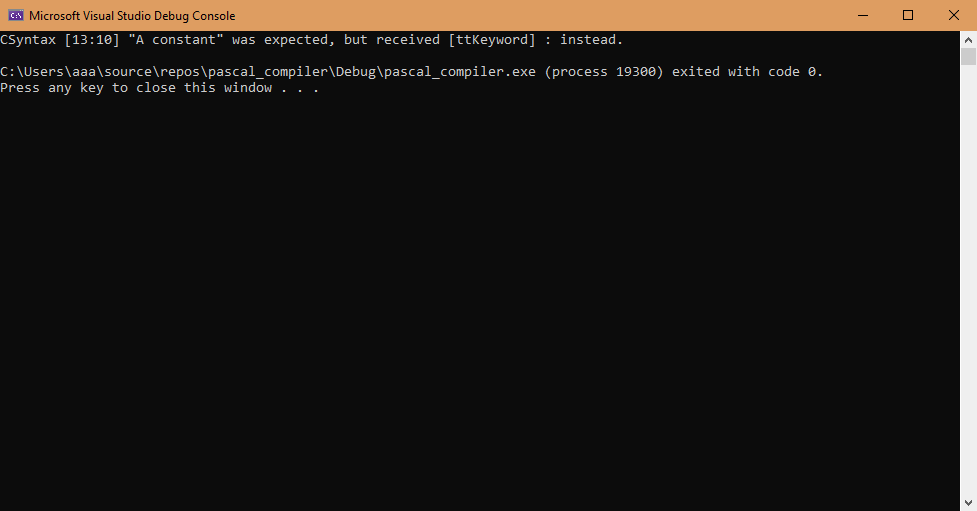
{true} : outpt := 'Fine';

false: outpt := 'Fine';

end;

end.

Вывод:



7 Семантический анализатор

7.1 Общее описание

Семантический анализатор отвечает за проверку правильности неформальных правил языка. Необходимо проверять следующие условия:

1. В любой области действия без внутренних по отношению к ней областей действия никакой идентификатор не может быть описан более одного раза;

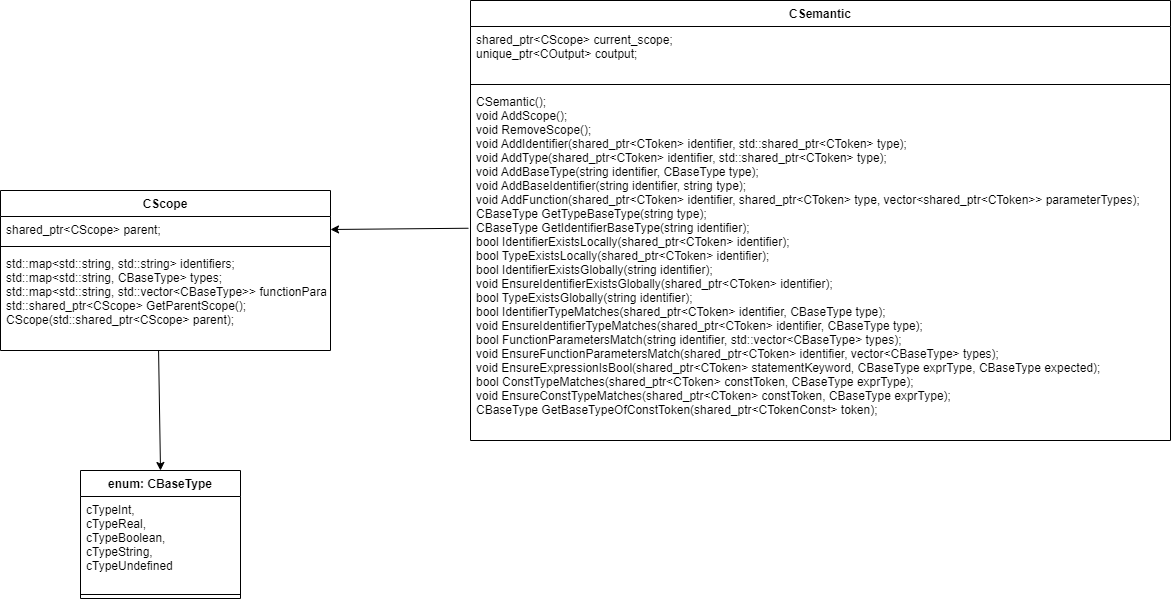
2. Каждому прикладному вхождению нестандартного идентификатора (стандартные идентификаторы – integer, boolean, real, true, false и др.) должно найтись соответствующее ему определяющее вхождение в текущей области видимости или выше;

3. Соответствия типов величин, входящих в синтаксические конструкции программ.

Учитывая условия выше нам необходимо следующее:

* Хранить информацию о типах;
* Хранить информацию о текущей области видимости

7.2 Структура семантического анализатора



Основой семантического анализатора является класс CSemantic, который содержит все методы для обработки ошибок. Данный класс поддерживает указатель на текущую область видимости. Данные об области видимости содержаться в объектах класса CScope: информация об идентификаторах, типах и параметрах функций.

Объект класса CSemantic является полем класса CSyntax, семантический анализ выполняется посредством взаимодействия с ним.

7.2 Ошибки семантического анализатора

CErrorSemanticAlreadyDefined – Идентификатор уже объявлен в локальной области действия

CErrorSemanticNotDefined – Идентификатор не объявлен

CErrorSemanticTypeMismatch – Попытка присвоить переменной неверный тип

CErrorSemanticWrongExprType – Неверный тип выражения (условия для if и while)

CErrorSemanticIncorrectParameters – Неверные параметры у вызова функции

CErrorSemanticCantTypeCast – Ошибка при неявном приведении типов в выражении

CErrorSemanticWrongConstType – Неверный тип константы (case)

* 1. Тестирование

Места, вызывающие ошибку, выделены красным.

1. Идентификатор уже объявлен в локальной области действия

program gcd;

type

Bptr = boolean;

var a,b,c,t:integer;

inpt, outpt: string;

d: Bptr;

e: boolean;

a,b, f,h: real;

{

this is a comment

}

function GCD(m,n:integer):integer;

type

w = boolean;

var ee, ee:integer;

function FFFF(m,n:integer):integer;

type

w1 = boolean;

var ee1, ee1:integer;

begin

while m<>n do

if m>n then m:=m-n else n:=n-m;

GCD:=m;

end;

begin

while m<>n do

if m>n then m:=m-n else n:=n-m;

GCD:=m;

end;

begin

inpt := 'Input 3 numbers';

e:= true;

d := e;

f := 123.4567;

a := (a+b)\*c - d/ ( GCD(64,18));

if (a = 30) and (b >= 15) or (c <= 17) xor (t <> 20) and not false then outpt := 'Fine';

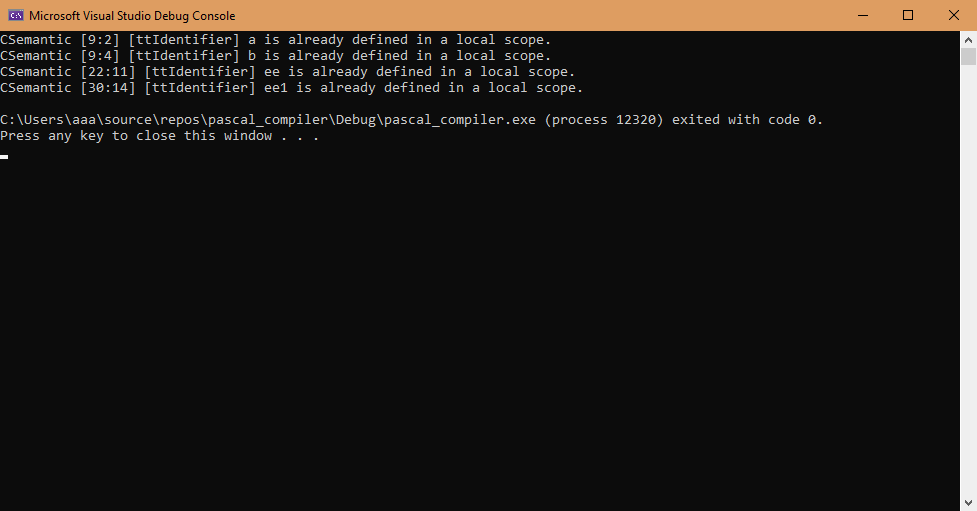
case a = 30 of

true : outpt := 'Fine';

false: outpt := 'Fine';

end;

end.



1. Идентификатор не объявлен

program gcd;

type

Bptr = boolean;

var a,b,c,t:integer;

inpt, outpt: string;

d: Bptr;

e: boolean;

h: real;

{

this is a comment

}

function GCD(m,n:integer):integer;

type

w = boolean;

var ee:integer;

function FFFF(m,n:integer):integer;

type

w1 = boolean;

var ee1:integer;

begin

while k<>i do

if m>n then m:=m-n else n:=n-m;

GCD:=m;

end;

begin

while m<>n do

if m>n then m:=m-n else n:=n-m;

GCD:=m;

end;

begin

inpt := 'Input 3 numbers';

e:= true;

d := e;

f := 123.4567;

a := (a+b)\*c - d/ ( GCD(64,18));

if (a = 30) and (b >= 15) or (c <= 17) xor (t <> 20) and not false then outpt := 'Fine';

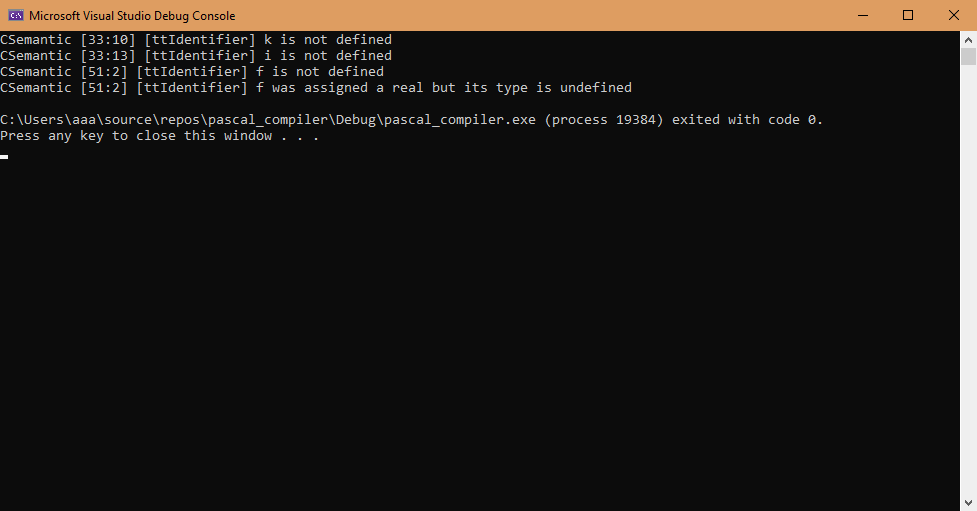
case a = 30 of

true : outpt := 'Fine';

false: outpt := 'Fine';

end;

end.



1. Попытка присвоить переменной неверный тип

program gcd;

type

Bptr = boolean;

var a,b,c, t:integer;

inpt, outpt: string;

d: Bptr;

e: boolean;

f,h: real;

{

this is a comment

}

function GCD(m,n:integer):integer;

type

w = boolean;

var ee:integer;

function FFFF(m,n:integer):integer;

type

w1 = boolean;

var ee1:integer;

begin

while m<>n do

if m>n then m:=m-n else n:=n-m;

GCD:=m;

end;

begin

while m<>n do

if m>n then m:=m-n else n:=n-m;

GCD:=m;

end;

begin

inpt := 'Input 3 numbers';

e:= true;

d := e;

f := 123.4567;

a := 52452.23125;

e := 'I am a boolean';

f := 123;

a := (a+b)\*c - d/ ( GCD(64,18));

if (a = 30) and (b >= 15) or (c <= 17) xor (t <> 20) and not false then outpt := 'Fine';

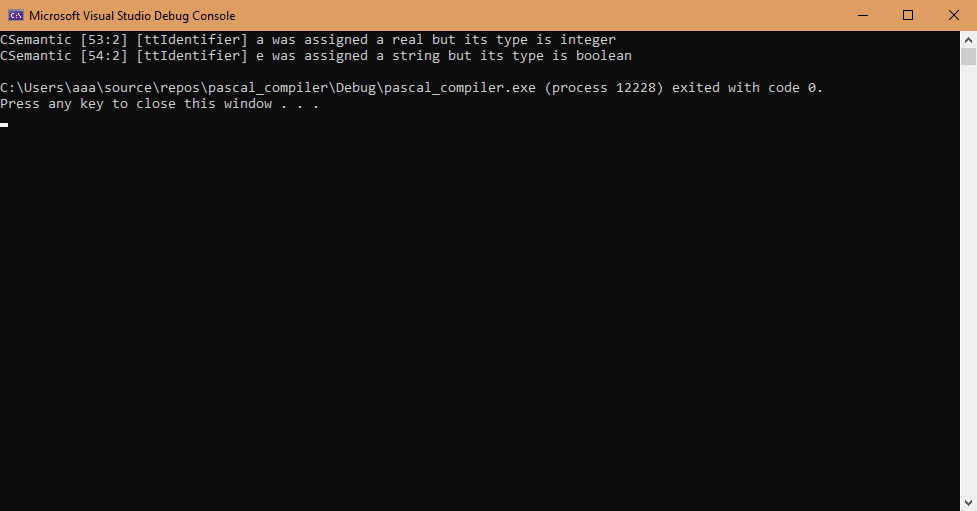
case a = 30 of

true : outpt := 'Fine';

false: outpt := 'Fine';

end;

end.



1. Неверный тип выражения (условия для if и while)

program gcd;

type

Bptr = boolean;

var a,b,c, t:integer;

inpt, outpt: string;

d: Bptr;

e: boolean;

f,h: real;

{

this is a comment

}

function GCD(m,n:integer):integer;

type

w = boolean;

var ee:integer;

function FFFF(m,n:integer):integer;

type

w1 = boolean;

var ee1:integer;

begin

while 1+2 do

if m>n then m:=m-n else n:=n-m;

GCD:=m;

end;

begin

while 23123.5135 do

if m>n then m:=m-n else n:=n-m;

GCD:=m;

end;

begin

inpt := 'Input 3 numbers';

e:= true;

d := e;

f := 123.4567;

a := (a+b)\*c - d/ ( GCD(64,18));

if 'I am a boolean' then outpt := 'Fine';

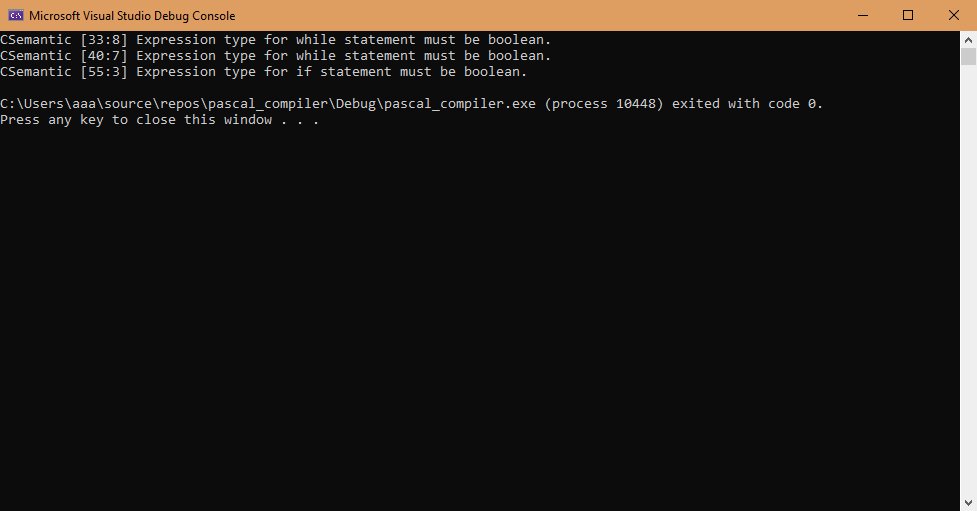
case a = 30 of

true : outpt := 'Fine';

false: outpt := 'Fine';

end;

end.



1. Неверные аргументы функции

program gcd;

type

Bptr = boolean;

var a,b,c, t:integer;

inpt, outpt: string;

d: Bptr;

e: boolean;

f,h: real;

{

this is a comment

}

function GCD(m,n:integer):integer;

type

w = boolean;

var ee:integer;

function FFFF(m,n:integer):integer;

type

w1 = boolean;

var ee1:integer;

begin

while m<>n do

if m>n then m:=m-n else n:=n-m;

GCD:=m;

end;

begin

while m<>n do

if m>n then m:=m-n else n:=n-m;

GCD:=FFFF(123);

end;

begin

inpt := 'Input 3 numbers';

e:= true;

d := e;

f := 123.4567;

a := (a+b)\*c - d/ ( GCD(123.123,'I am an integer'));

b := GCD();

if (a = 30) and (b >= 15) or (c <= 17) xor (t <> 20) and not false then outpt := 'Fine';

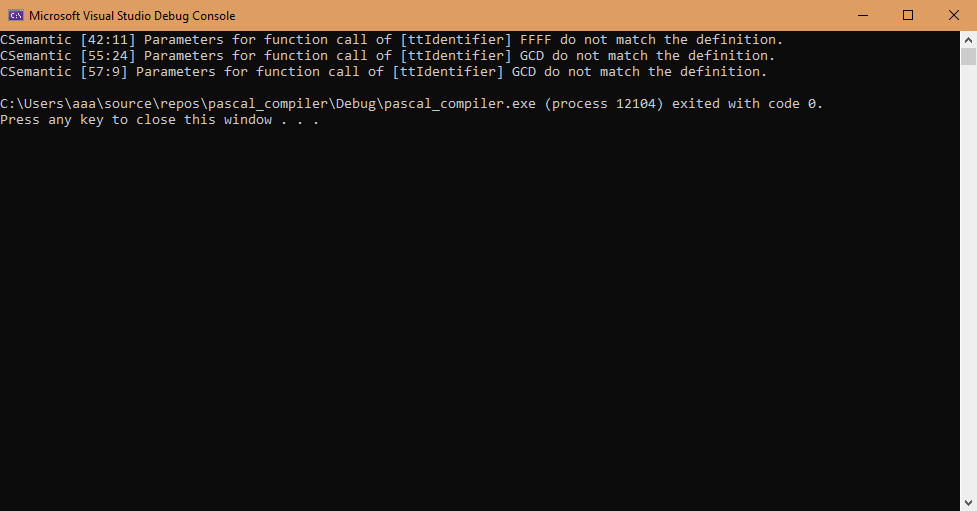
case a = 30 of

true : outpt := 'Fine';

false: outpt := 'Fine';

end;

end.



1. Ошибка при неявном приведении типов в выражении

program gcd;

type

Bptr = boolean;

var a,b,c, t:integer;

inpt, outpt: string;

d: Bptr;

e: boolean;

f,h: real;

{

this is a comment

}

function GCD(m,n:integer):integer;

type

w = boolean;

var ee:integer;

function FFFF(m,n:integer):integer;

type

w1 = boolean;

var ee1:integer;

begin

while m<>n do

if m>n then m:=m-n else n:=n-m;

GCD:=m;

end;

begin

while m<>n do

if m>n then m:=m-n else n:=n-m;

GCD:=m;

end;

begin

inpt := 'Input 3 numbers';

e:= true;

d := e;

f := 123.4567 + '123';

a := (a+b)\*c - d/ ( GCD(64,18));

if 15 and (b >= 15) or (c <= 'Hi') xor (t <> 'I am a number') and not 12 then outpt := 'Fine';

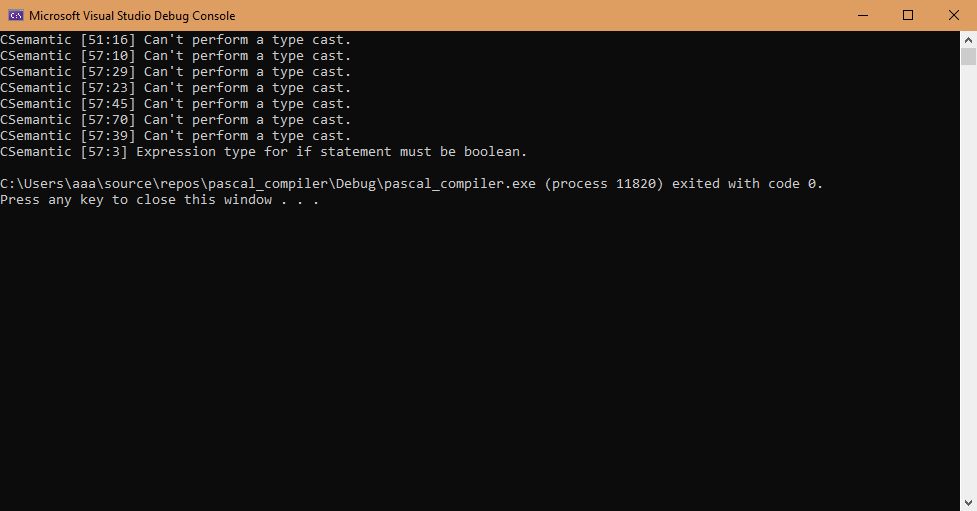
case a = 30 of

true : outpt := 'Fine';

false: outpt := 'Fine';

end;

end.



1. Неверный тип константы

program gcd;

type

Bptr = boolean;

var a,b,c, t:integer;

inpt, outpt: string;

d: Bptr;

e: boolean;

f,h: real;

{

this is a comment

}

function GCD(m,n:integer):integer;

type

w = boolean;

var ee:integer;

function FFFF(m,n:integer):integer;

type

w1 = boolean;

var ee1:integer;

begin

while m<>n do

if m>n then m:=m-n else n:=n-m;

GCD:=m;

end;

begin

while m<>n do

if m>n then m:=m-n else n:=n-m;

GCD:=m;

end;

begin

inpt := 'Input 3 numbers';

e:= true;

d := e;

f := 123.4567;

a := (a+b)\*c - d/ ( GCD(64,18));

if (a = 30) and (b >= 15) or (c <= 17) xor (t <> 20) and not false then outpt := 'Fine';

case a = 30 of

1 : outpt := 'Fine';

2: outpt := 'Fine';

end;

case a of

'one' : f := 1;

'two': f := 2;

end;

end.

