|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ В РФ | | | | | | | | | |
|  | | | |  | | |  | | |
|  | | ФГБОУ ВО «Пермский государственный национальный исследовательский университет» | | | | | |  | |
|  | | | |  | | |  | | |
|  | | Отчёт  по дисциплине «Формальные грамматики и методы трансляции» | | | | | |  | |
|  | | |  | | | |  | | |
|  | Работу выполнил  студент гр. ПМИ-4-18  Колесников А.С. \_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2022 | | | |  | Проверил  ассистент кафедры МОВС  Пономарев Ф.А.\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2022 | | |  |
|  | | | | | | | | | |
| Пермь 2022 | | | | | | | | | |

Оглавление

[1 Структура компилятора 3](#_Toc94468823)

[2 Модуль ввода-вывода 4](#_Toc94468824)

[3 Лексический анализатор 5](#_Toc94468825)

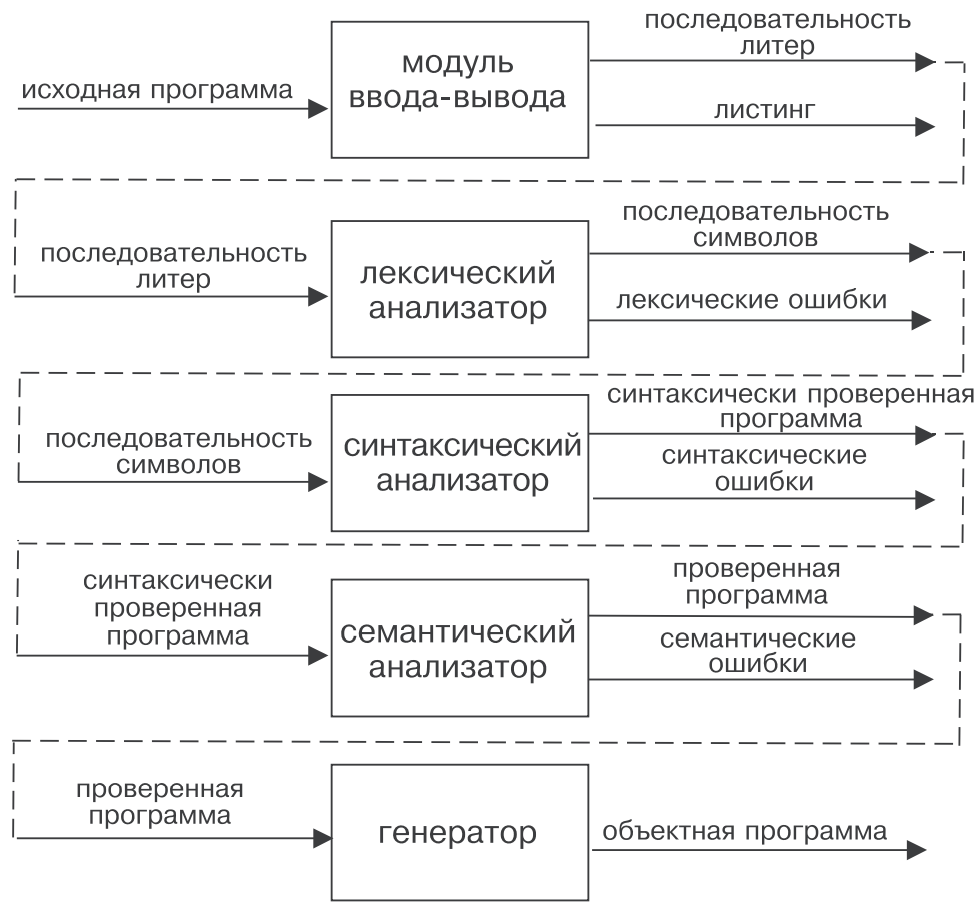
[4 Синтаксический анализатор 6](#_Toc94468826)

[5 Семантический анализатор 7](#_Toc94468827)

1. Структура компилятора

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

Разработка программного обеспечения подразумевает модульность и хорошую структурированность программ. Учитывая это, представим компилятор как совокупность логически взаимосвязанных модулей (рис. 1).



Рисунок

1 – Структура компилятора

Модуль ввода вывода отвечает за все действия, связанные с чтением и записью данных. Для каждого типа устройства может существовать свой собственный модуль ввода-вывода.

Лексический анализатор осуществляет простой лексический анализ исходной программы. На вход он получает последовательность литер, а отдает последовательность лексем (токенов).

Синтаксический анализатор проверяет, удовлетворяет ли программа формальным правилам.

Семантический анализатор проверяет не нарушены ли неформальные правила описания языка.

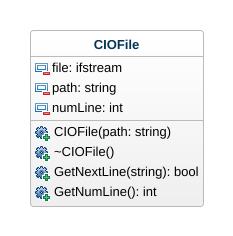
Генератор отображает независимое промежуточное представление исходной программы на реальную ЭВМ, поэтому должен переписываться для каждой новой машины.

1. Модуль ввода-вывода
   1. Проектирование

Модуль ввода-вывода считывает последовательность литер исходной программы с внешнего устройства и передает их лексическому анализатору, который проверяет, удовлетворят ли эта последовательность литер правилам описания языка, и формирует в случае необходимости сообщения об ошибках.

Будем считать, что в результате очередного обращения к модулю ввода-вывода анализатор получает текущую строку.

Для печати сообщения об ошибке, лексический анализатор должен передать модулю ввода-вывода причину и местоположение ошибки в строке.



Рисунок

2 – Структура класса модуля ввода-вывода

* 1. Тестирование

Проверим работоспособность считывание следующего файла:

var

a,b,c:integer;

name:string;

x:real;

flag:boolean;

type

myint = integer;

str=string;

begin

//some comment

a:= 10 mod 2;

b := 6 div 4;

x:= 2.1;

name :='some string';

flag:= True;

if (a = 2) and (b < 1) then

if (x >= 0) or (name <> 'some str') then

c := 2 + 4

else

name := 'text';

while 2 >= 5 do

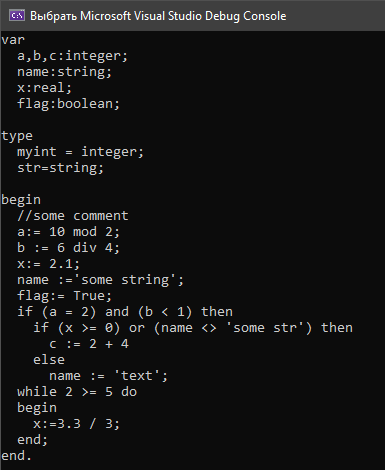
begin

x:=3.3 / 3;

end;

end.

Результат представлен на рисунке:



Рисунок

– Результат работы модуля ввода-вывода

1. Лексический анализатор
   1. Проектирование

Лексический анализатор формирует токены исходной программы и строит их внутреннее представление, а также распознает и исключает комментарии, которые не нужны для дальнейшей трансляции.

Лексическим анализатор запрашивает у модуля ввода-вывода следующую строку и обрабатывает ее, выделяя все лексемы. Если же файл был обработан полностью, то модуль ввода-вывода, уведомляет об этом лексический анализатор, и последний завершает работу.

Рассмотрим внутреннее представление токенов.

Лексический анализатор преобразует последовательности литер переменной длины в токены. Всего их три типа.

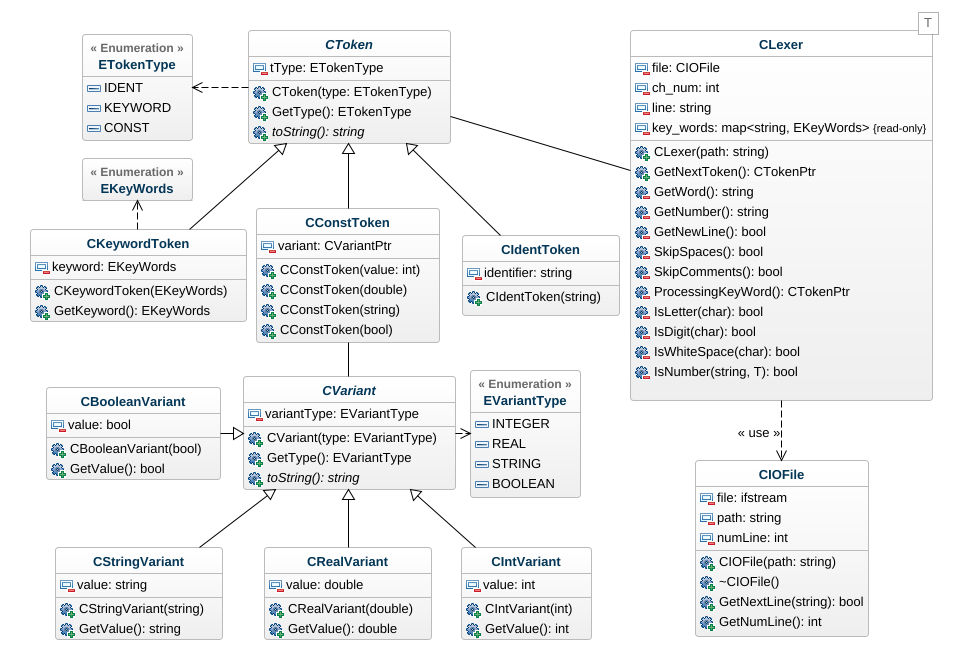
Константы представляют определенный тип данных. В нашем случае могут быть 4 вида константных токенов: integer, real, boolean, string.

Ключевые слова представляют последовательности символов зарезервированные для использования при написании программы. Они представлены в виде перечислений (enum).

Пользовательские идентификаторы — это последовательности символов, содержащие цифры, буквы и знаки подчеркивания, которые может использовать пользователь для обозначения переменных, функции, собственных типов и т.д.

При обнаружении ошибки на этапе анализа программы необходимо напечатать соответствующее сообщение и указать место, где обнаружена ошибка. Поэтому лексический анализатор должен формировать координаты каждого символа, а именно: номер строки исходной программы и позицию символа (позицию первой литеры символа) в этой строке.

На рисунке 3 представлена диаграмма классов, которые использует лексический анализатор для построения внутренней модели.



Рисунок

4 – Диаграмма классов лексического анализатора

Класс CToken базовый абстрактный класс для всех токенов.

Перечисление ETokenType содержит в себе все типы возвращаемых токенов классом CToken.

Класс CKeywordToken производный от CToken и отвечает за ключевые слова. Содержит тип ключевого слова, который представлен перечислением EKeyWords.

Класс CIdentToken производный от CToken и отвечает за хранения пользовательских идентификаторов.

Класс CConstToken производный от CToken и отвечает за хранения констант. Содержит указатель на CVariant.

Класс CVariant базовый абстрактный класс для типов данных. Хранит тип данных, представленный в виде перечисления EVariantType.

Класс CBooleanVariant отвечает за хранение логических значений типа boolean.

Класс CStringVariant отвечает за хранение строковых значений типа string.

Класс CRealVariant отвечает за хранение чисел с плавающей точкой типа real.

Класс CIntVariant отвечает за хранение целочисленных значений типа integer.

Класс CLexer отвечает за получение токенов по средством функции GetNextToken, которая возвращает указатель на экземпляр класса CToken, а также уведомляет об ошибках лексического уровня. Для считывания файла и выдачи листинга, данный класс использует модуль ввода-вывода CIOFile.

* 1. Тестирование

Для того чтоб протестировать работу лексического анализатора проверим его на нескольких программах.

Первая программа

Ниже представлена текст программы, которую нужно обработать:

var

a,b,c:integer;

name:string;

x:real;

flag:boolean;

begin

// some comment here

a:= 10 - 3;

b := 2 \*(3 + 4);

c:=a\*b;

x:= 5.65 /2.1;

{ and

here

}

name :='some string';

flag:= True;

flag := False;

(\* the end of

the program \*)

end.

В итоге были получены следующие результаты

1: KEYWORD: var

2: IDENT: a

3: KEYWORD: ,

4: IDENT: b

5: KEYWORD: ,

6: IDENT: c

7: KEYWORD: :

8: KEYWORD: integer

9: KEYWORD: ;

10: IDENT: name

11: KEYWORD: :

12: KEYWORD: string

13: KEYWORD: ;

14: IDENT: x

15: KEYWORD: :

16: KEYWORD: real

17: KEYWORD: ;

18: IDENT: flag

19: KEYWORD: :

20: KEYWORD: boolean

21: KEYWORD: ;

22: KEYWORD: begin

23: IDENT: a

24: KEYWORD: :=

25: CONST: integer: 10

26: KEYWORD: -

27: CONST: integer: 3

28: KEYWORD: ;

29: IDENT: b

30: KEYWORD: :=

31: CONST: integer: 2

32: KEYWORD: \*

33: KEYWORD: (

34: CONST: integer: 3

35: KEYWORD: +

36: CONST: integer: 4

37: KEYWORD: )

38: KEYWORD: ;

39: IDENT: c

40: KEYWORD: :=

41: IDENT: a

42: KEYWORD: \*

43: IDENT: b

44: KEYWORD: ;

45: IDENT: x

46: KEYWORD: :=

47: CONST: real: 5.650000

48: KEYWORD: /

49: CONST: real: 2.100000

50: KEYWORD: ;

51: IDENT: name

52: KEYWORD: :=

53: CONST: string: some string

54: KEYWORD: ;

55: IDENT: flag

56: KEYWORD: :=

57: CONST: boolean: True

58: KEYWORD: ;

59: IDENT: flag

60: KEYWORD: :=

61: CONST: boolean: False

62: KEYWORD: ;

63: KEYWORD: end

64: KEYWORD: .

Вторая программа

Ниже представлена текст программы, которую нужно обработать:

var

a,b,c:integer;

name:string;

x:real;

flag:boolean;

type

myint = integer;

str=string;

begin

//some comment

a:= 10 mod 2;

b := 6 div 4;

x:= 2.1;

name :='some string';

flag:= True;

if (a = 2) and (b < 1) then

if (x >= 0) or (name <> 'some str') then

c := 2 + 4

else

name := 'text';

while 2 >= 5 do

begin

x:=3.3 / 3;

end;

end.

В итоге были получены следующие результаты

1: KEYWORD: var

2: IDENT: a

3: KEYWORD: ,

4: IDENT: b

5: KEYWORD: ,

6: IDENT: c

7: KEYWORD: :

8: KEYWORD: integer

9: KEYWORD: ;

10: IDENT: name

11: KEYWORD: :

12: KEYWORD: string

13: KEYWORD: ;

14: IDENT: x

15: KEYWORD: :

16: KEYWORD: real

17: KEYWORD: ;

18: IDENT: flag

19: KEYWORD: :

20: KEYWORD: boolean

21: KEYWORD: ;

22: KEYWORD: type

23: IDENT: myint

24: KEYWORD: =

25: KEYWORD: integer

26: KEYWORD: ;

27: IDENT: str

28: KEYWORD: =

29: KEYWORD: string

30: KEYWORD: ;

31: KEYWORD: begin

32: IDENT: a

33: KEYWORD: :=

34: CONST: integer: 10

35: IDENT: mod

36: CONST: integer: 2

37: KEYWORD: ;

38: IDENT: b

39: KEYWORD: :=

40: CONST: integer: 6

41: IDENT: div

42: CONST: integer: 4

43: KEYWORD: ;

44: IDENT: x

45: KEYWORD: :=

46: CONST: real: 2.100000

47: KEYWORD: ;

48: IDENT: name

49: KEYWORD: :=

50: CONST: string: some string

51: KEYWORD: ;

52: IDENT: flag

53: KEYWORD: :=

54: CONST: boolean: True

55: KEYWORD: ;

56: KEYWORD: if

57: KEYWORD: (

58: IDENT: a

59: KEYWORD: =

60: CONST: integer: 2

61: KEYWORD: )

62: KEYWORD: and

63: KEYWORD: (

64: IDENT: b

65: KEYWORD: <

66: CONST: integer: 1

67: KEYWORD: )

68: KEYWORD: then

69: KEYWORD: if

70: KEYWORD: (

71: IDENT: x

72: KEYWORD: >=

73: CONST: integer: 0

74: KEYWORD: )

75: KEYWORD: or

76: KEYWORD: (

77: IDENT: name

78: KEYWORD: <>

79: CONST: string: some str

80: KEYWORD: )

81: KEYWORD: then

82: IDENT: c

83: KEYWORD: :=

84: CONST: integer: 2

85: KEYWORD: +

86: CONST: integer: 4

87: KEYWORD: else

88: IDENT: name

89: KEYWORD: :=

90: CONST: string: text

91: KEYWORD: ;

92: KEYWORD: while

93: CONST: integer: 2

94: KEYWORD: >=

95: CONST: integer: 5

96: KEYWORD: do

97: KEYWORD: begin

98: IDENT: x

99: KEYWORD: :=

100: CONST: real: 3.300000

101: KEYWORD: /

102: CONST: integer: 3

103: KEYWORD: ;

104: KEYWORD: end

105: KEYWORD: ;

106: KEYWORD: end

107: KEYWORD: .

Третья программа

Ниже представлена текст программы, которую нужно обработать:

var x,y,z:real;

a:integer;

b:integer;

function Sum(a,b: real): real;

begin

Result := a + b;

end;

begin

x:=0.5;

y := 1.4005;

z :=Sum(x,y);

for var i := 1 to 10 do

a := a + i;

repeat

b:= b + 1;

until b > 10;

end.

В итоге были получены следующие результаты

1: KEYWORD: var

2: IDENT: x

3: KEYWORD: ,

4: IDENT: y

5: KEYWORD: ,

6: IDENT: z

7: KEYWORD: :

8: KEYWORD: real

9: KEYWORD: ;

10: IDENT: a

11: KEYWORD: :

12: KEYWORD: integer

13: KEYWORD: ;

14: IDENT: b

15: KEYWORD: :

16: KEYWORD: integer

17: KEYWORD: ;

18: KEYWORD: function

19: IDENT: Sum

20: KEYWORD: (

21: IDENT: a

22: KEYWORD: ,

23: IDENT: b

24: KEYWORD: :

25: KEYWORD: real

26: KEYWORD: )

27: KEYWORD: :

28: KEYWORD: real

29: KEYWORD: ;

30: KEYWORD: begin

31: IDENT: Result

32: KEYWORD: :=

33: IDENT: a

34: KEYWORD: +

35: IDENT: b

36: KEYWORD: ;

37: KEYWORD: end

38: KEYWORD: ;

39: KEYWORD: begin

40: IDENT: x

41: KEYWORD: :=

42: CONST: real: 0.500000

43: KEYWORD: ;

44: IDENT: y

45: KEYWORD: :=

46: CONST: real: 1.400500

47: KEYWORD: ;

48: IDENT: z

49: KEYWORD: :=

50: IDENT: Sum

51: KEYWORD: (

52: IDENT: x

53: KEYWORD: ,

54: IDENT: y

55: KEYWORD: )

56: KEYWORD: ;

57: KEYWORD: for

58: KEYWORD: var

59: IDENT: i

60: KEYWORD: :=

61: CONST: integer: 1

62: KEYWORD: to

63: CONST: integer: 10

64: KEYWORD: do

65: IDENT: a

66: KEYWORD: :=

67: IDENT: a

68: KEYWORD: +

69: IDENT: i

70: KEYWORD: ;

71: KEYWORD: repeat

72: IDENT: b

73: KEYWORD: :=

74: IDENT: b

75: KEYWORD: +

76: CONST: integer: 1

77: KEYWORD: ;

78: KEYWORD: until

79: IDENT: b

80: KEYWORD: >

81: CONST: integer: 10

82: KEYWORD: ;

83: KEYWORD: end

84: KEYWORD: .

1. Синтаксический анализатор
   1. Проектирование

Синтаксический анализатор по мере необходимости запрашивает у модуля ввода вывода следующий токен.

* 1. Тестирование

1. Семантический анализатор