|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ | | | | | |  | | | |
|  | | | |  | | | | | | | |
|  | Федеральное государственное автономное образовательное учреждение высшего образования  «Пермский государственный национальный исследовательский университет» | | | | | | | | | |  |
|  | | | | Кафедра математического обеспечения вычислительных систем | | | | | | | |
|  | **Отчет**  **По разработке компилятора языка Pascal** | | | | | | | |  | | |
|  | | |  | | | |  | | | | |
|  | |  | | |  | Работу выполнил студент группы ПМИ-3 4 курса механико-математического факультета  \_\_\_\_\_\_\_\_А.А. Солдатов | | | |  | |
|  | |  | | |  | Научный руководитель:  канд. физ.-мат. наук,  доцент каф. МОВС  \_\_\_\_\_\_\_\_Ф.А. Пономарев  “\_\_\_” \_\_\_\_\_\_\_\_\_\_2022 г. | | | |  | |
|  | | | Пермь 2022 | | | |  | | | | |

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc102006834)

[2 Структура компилятора 4](#_Toc102006835)

[3 Модуль ввода-вывода 5](#_Toc102006836)

[4 Лексический анализатор 6](#_Toc102006837)

[5 Синтаксический анализатор 7](#_Toc102006838)

[6 Семантический анализатор 8](#_Toc102006839)

# 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.

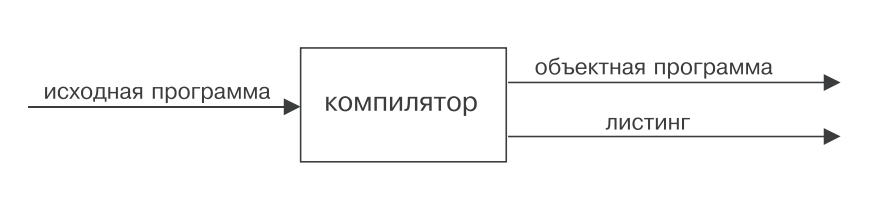
Замена задания по генерации на теоретический экзамен возможна также в случае, если студента не устраивают баллы, полученные за задание по генерации. В этом случае набранные баллы обнуляются, и студент сдает экзамен на тех же условиях, что и студенты, не сдававшие задание по генерации кода вообще.

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.

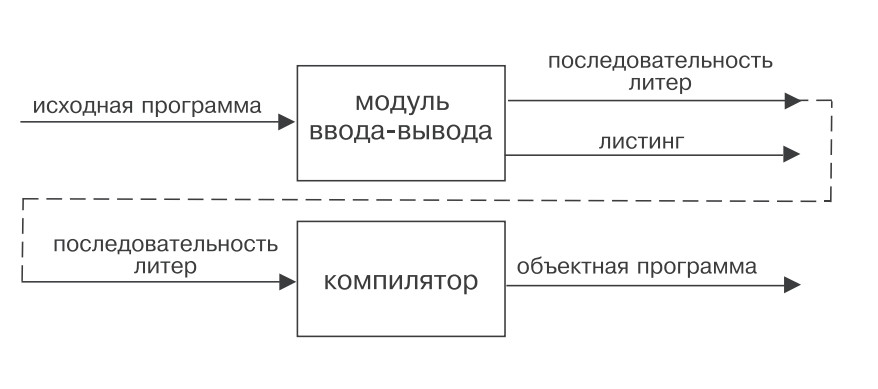
Общая дополнительная часть. Раздел описания типов. Выражение (полностью, включая арифметические, логические операции, сравнения и т.д., но только над константами и простыми переменными (не индексированные, не поля записи, не указатели)). Условный оператор (if). Оператор цикла с предусловием (while).

# 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. Схема компилятора:



При вводе исходной программы и получении листинга мы имеем дело с конкретными устройствами ввода-вывода (клавиатура, дисплей, магнитные диски). Чтобы легко адаптировать компилятор к различным внешним устройствам конкретной машины, отделим все действия по вводу-выводу данных от собственно процесса компиляции:

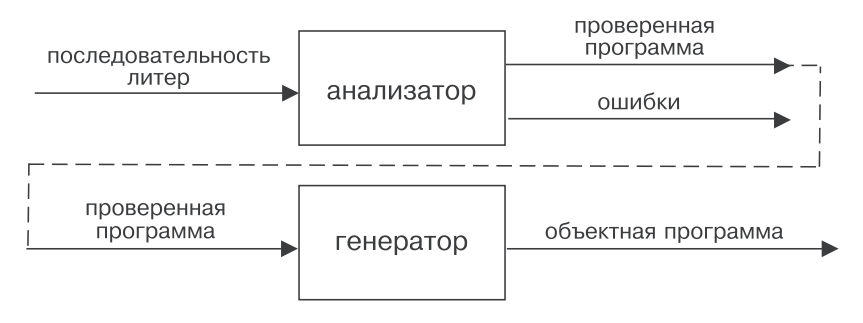


Работа компилятора включает в себя два основных этапа:

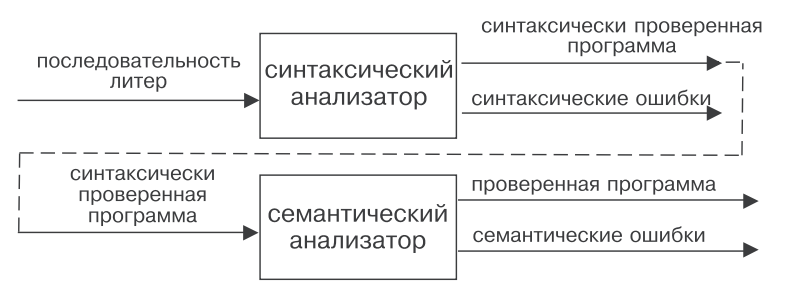
1) анализ — определение правильности исходной программы и формирование (в случае необходимости) сообщений об ошибках;

2) синтез — генерация объектной программы; этот этап выполняется для программ, не содержащих ошибок.

Таким образом, собственно компилятор разбивается на составляющие модули:



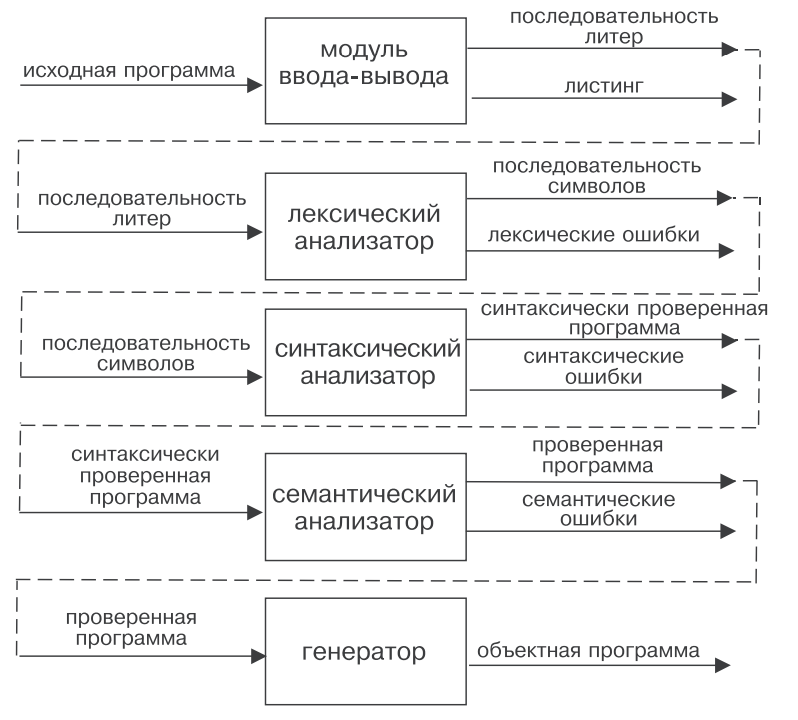
Учитывая особенности описания синтаксиса языков программирования, разделим анализатор на два модуля:



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 обычно выполняется внутри синтаксического анализатора. Первый модуль – лексический анализатор, просматривает текст (последовательность литер) исходной программы и строит символы (лексемы) — идентификаторы, ключевые слова, разделители, числа. Второй модуль (синтаксический анализатор) выполняет синтаксический анализ последовательности символов. На этом этапе символы рассматриваются как неделимые, и их представление как последовательности литер несущественно.

Итак, в результате мы получили следующую структуру компилятора:



# 3 Модуль ввода-вывода

## 3.1 Ввод

Объявим класс для ввода данных:

class Reader {

public:

Reader(istream& stream);

~Reader();

pair<char, pair<int, int>> getCh();

private:

istream& stream;

int line, index;

};

В конструктор класса передается ссылка на поток ввода, чтобы можно было использовать различные источники ввода.

Также будем хранить номер текущей строки и номер текущего символа в строке в line и index соответственно.

Метод getCh возвращает пару из следующего символа и пары с номером текущей строки и номером текущего символа.

### Тестирование

Протестируем работу модуля на файле с программой на Паскале:

var

r1,r2: integer; // values on cubes

begin

r1 := Random(6)+1;

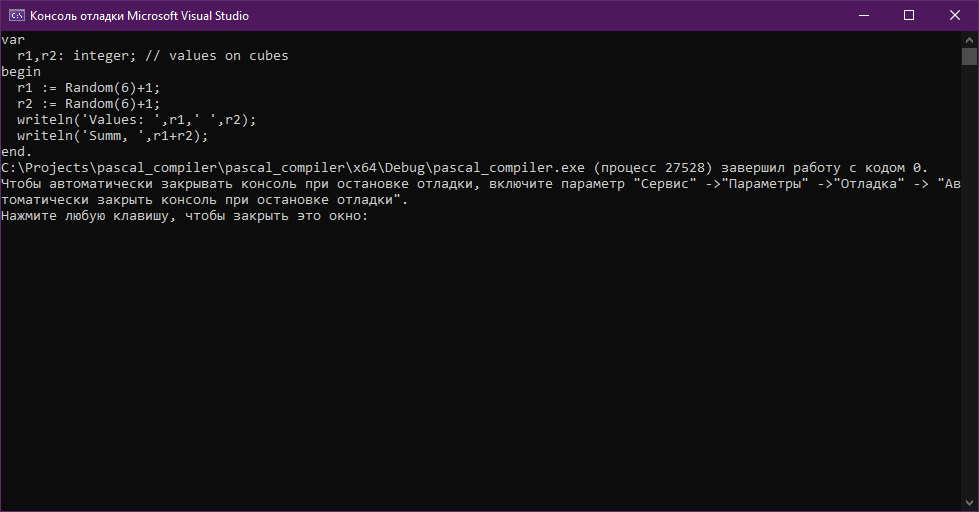
r2 := Random(6)+1;

writeln('Values: ',r1,' ',r2);

writeln('Summ, ',r1+r2);

end.

Вывод:



Теперь выведем символы вместе с их позициями:

v 0 0

a 0 1

r 0 2

0 3

0 4

1 0

1 1

r 1 2

1 1 3

, 1 4

r 1 5

2 1 6

: 1 7

1 8

i 1 9

n 1 10

t 1 11

e 1 12

g 1 13

e 1 14

r 1 15

; 1 16

1 17

/ 1 18

/ 1 19

1 20

v 1 21

a 1 22

l 1 23

u 1 24

e 1 25

s 1 26

1 27

o 1 28

n 1 29

1 30

c 1 31

u 1 32

b 1 33

e 1 34

s 1 35

1 36

b 2 0

e 2 1

g 2 2

i 2 3

n 2 4

2 5

3 0

3 1

r 3 2

1 3 3

3 4

: 3 5

= 3 6

3 7

R 3 8

a 3 9

n 3 10

d 3 11

o 3 12

m 3 13

( 3 14

6 3 15

) 3 16

+ 3 17

1 3 18

; 3 19

3 20

4 0

4 1

r 4 2

2 4 3

4 4

: 4 5

= 4 6

4 7

R 4 8

a 4 9

n 4 10

d 4 11

o 4 12

m 4 13

( 4 14

6 4 15

) 4 16

+ 4 17

1 4 18

; 4 19

4 20

5 0

5 1

w 5 2

r 5 3

i 5 4

t 5 5

e 5 6

l 5 7

n 5 8

( 5 9

' 5 10

V 5 11

a 5 12

l 5 13

u 5 14

e 5 15

s 5 16

: 5 17

5 18

' 5 19

, 5 20

r 5 21

1 5 22

, 5 23

' 5 24

5 25

' 5 26

, 5 27

r 5 28

2 5 29

) 5 30

; 5 31

5 32

6 0

6 1

w 6 2

r 6 3

i 6 4

t 6 5

e 6 6

l 6 7

n 6 8

( 6 9

' 6 10

S 6 11

u 6 12

m 6 13

m 6 14

, 6 15

6 16

' 6 17

, 6 18

r 6 19

1 6 20

+ 6 21

r 6 22

2 6 23

) 6 24

; 6 25

6 26

e 7 0

n 7 1

d 7 2

. 7 3

## 3.2 Вывод

Объявим класс для вывода данных:

class Writer {

public:

Writer(ostream& stream);

~Writer();

/\*template<class T> friend unique\_ptr<Writer>& operator << (const unique\_ptr<Writer>&,const T& t);\*/

ostream& stream;

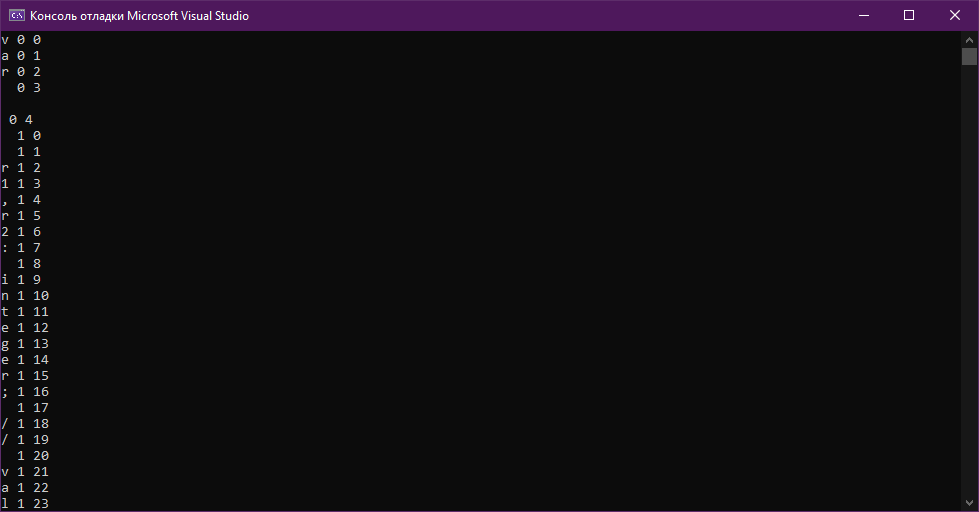
};

В конструктор передается поток вывода.

У меня не получилось адекватно перегрузить оператор «<<» для удобной работы с выводом, поэтому для вывода просто обращаемся к публичному полю stream.

### Тестирование

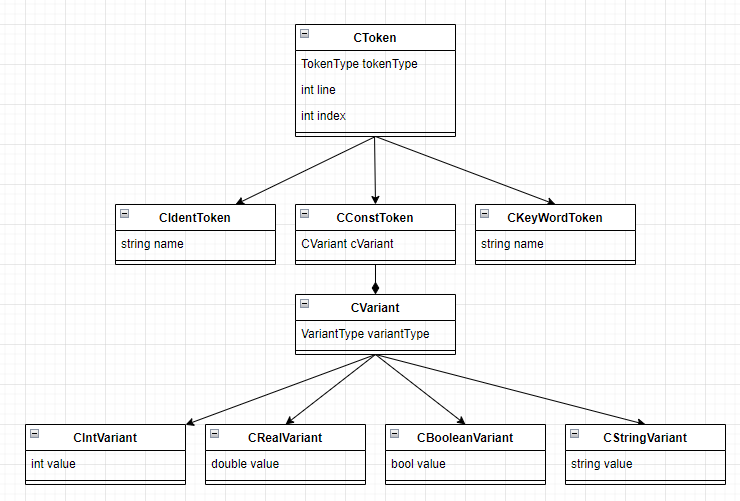
Протестируем модуль с той же программой, что в модуле ввода.



# 4 Лексический анализатор

Лексический анализатор используется для разбиения программы на лексемы/токены. В ходе работы он также проверяет программу на лексические ошибки. Комментарии игнорируются анализатором.

Токены могут быть 3 видов: идентификаторы, константы и ключевые слова языка. Константы может иметь один из 4х типов: integer, real, string или boolean (В нашей реализации компилятора будут поддерживаться только эти типы). Диаграмма классов для токенов:



TokenType и VariantType – перечислимые типы:

enum class VariantType {

vtInt,

vtReal,

vtString,

vtBoolean

};

enum class TokenType {

ttIdent,

ttKeyWord,

ttConst

};

Так же определим перечислимый тип для ключевых слов языка Pascal:

enum class CKeyWords {

programSy, // program

beginSy, // begin

endSy, // end

varSy, // var

plusSy, // +

minusSy, // -

multiplySy, // \*

divisionSy, // /

leftBracketSy, // (

rightBracketSy, // )

assignSy, // :=

dotSy, // .

commaSy, // ,

colonSy, // :

semicolonSy, // ;

eqSy, // =

ltSy, // <

gtSy, // >

leSy, // <=

geSy, // >=

neSy, // <>

ifSy, // if

thenSy, // then

elseSy, // else

whileSy, // while

doSy, // do

andSy, // and

orSy, // or

xorSy, // xor

notSy, // not

startCommentSy, // {

endCommentSy, // }

};

Для каждого перечислимого типа определим словарь, позволяющий по типу получить его строковое представление, а для ключевых слов определим и словарь для обратного преобразования из строки в тип.

Определим класс лексического анализатора:

class CLexer {

public:

CLexer(Reader\* reader);

unique\_ptr<CToken> getNextToken();

private:

unique\_ptr<Reader> reader;

char ch;

int line, index;

void skipComment();

pair<VariantType, string> getNumber();

string getFull();

string getString();

bool isLetter(char ch);

bool checkInt(string);

bool isValidNumber(string& number, int& separators);

void getNextChar();

};

В конструктор лексического анализатора передается модуль ввода. Лексический анализатор хранит текущий просматриваемый символ, текущий номер строки и номер символа. Основной метод getNextToken возвращает токен. Методы:

* skipComment() – пропускает комментарий в тексте программы
* getNumber() – возвращает тип числа и его строковую запись
* getFull() – возвращает имя идентификатора или ключевого слова
* getString() – возвращает содержимое строки
* isLetter() – проверяет, является ли символ буквой латинского алфавита или «\_»
* checkInt() – проверяет, поместится ли целое число в integer
* isValidNumber() – проверяет, является ли число целым или дробным
* getNextChar() – обновляет текущий символ, номер строки и символа, считывая их из модуля ввода

Будем обрабатывать следующие ошибки: неизвестный символ(символа нет в языке Pascal и он не является часть строки), непредвиденный символ(неверная последовательность символов), слишком большое число(для integer), неверная запись числа (в числе встретилось больше одной точки).

### Тестирование

Проверим вывод токенов на программе:

var

r1,r2: integer; { values on cubes }

begin

r1 := Random(6)+1;

r2 := Random(6)+1;

writeln('Values: ',r1,' ',r2);

writeln('Summ, ',r1+r2);

end.

Выведем токены с их типами:

0:0 var ttKeyWord

1:2 r1 ttIdent

1:4 , ttKeyWord

1:5 r2 ttIdent

1:7 : ttKeyWord

1:9 integer ttIdent

1:16 ; ttKeyWord

2:0 begin ttKeyWord

3:2 r1 ttIdent

3:5 := ttKeyWord

3:8 Random ttIdent

3:14 ( ttKeyWord

3:15 "6" integer ttConst

3:16 ) ttKeyWord

3:17 + ttKeyWord

3:18 "1" integer ttConst

3:19 ; ttKeyWord

4:2 r2 ttIdent

4:5 := ttKeyWord

4:8 Random ttIdent

4:14 ( ttKeyWord

4:15 "6" integer ttConst

4:16 ) ttKeyWord

4:17 + ttKeyWord

4:18 "1" integer ttConst

4:19 ; ttKeyWord

5:2 writeln ttIdent

5:9 ( ttKeyWord

5:10 "Values: " string ttConst

5:20 , ttKeyWord

5:21 r1 ttIdent

5:23 , ttKeyWord

5:24 " " string ttConst

5:27 , ttKeyWord

5:28 r2 ttIdent

5:30 ) ttKeyWord

5:31 ; ttKeyWord

6:2 writeln ttIdent

6:9 ( ttKeyWord

6:10 "Summ, " string ttConst

6:18 , ttKeyWord

6:19 r1 ttIdent

6:21 + ttKeyWord

6:22 r2 ttIdent

6:24 ) ttKeyWord

6:25 ; ttKeyWord

7:0 end ttKeyWord

7:3 . ttKeyWord

Проверим обработку ошибок на программе:

var

r1,r2: integer; { values on cubes }

r3 : real;

begin

r1 := 11111111111111111111111111111111111111;

r3 := 12341.12.12

r2 := %; {

end.

Вывод:

0:0 var ttKeyWord

1:2 r1 ttIdent

1:4 , ttKeyWord

1:5 r2 ttIdent

1:7 : ttKeyWord

1:9 integer ttIdent

1:16 ; ttKeyWord

2:2 r3 ttIdent

2:5 : ttKeyWord

2:7 real ttIdent

2:11 ; ttKeyWord

3:0 begin ttKeyWord

4:2 r1 ttIdent

4:5 := ttKeyWord

Error: 4:8 1003 IntegerLimit'11111111111111111111111111111111111111'

4:46 ; ttKeyWord

5:2 r3 ttIdent

5:5 := ttKeyWord

Error: 5:8 1002 InvalidNumber'12341.12.12'

6:2 r2 ttIdent

6:5 := ttKeyWord

Error: 6:8 1001 UnknownSymbol'%'

6:9 ; ttKeyWord

Error: 6:11 1000 UnexpectedSymbol'{'

# 5 Синтаксический анализатор

# 6 Семантический анализатор