|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ | | | | | | | | | | | | |
|  | | | | | |  | | |  | | | |
| ФГБОУ ВО «Пермский государственный национальный исследовательский университет» | | | | | | | | | | | | |
|  | | | | |  | | | | |  | | |
|  | | ОТЧЕТ  По разработке компилятора для языка Pascal  Вариант 11 | | | | | | | | |  | |
|  | | | |  | | | | | |  | | |
|  | Работу выполнил  студент гр. ПМИ-3  Субботин И.В.  (подпись)  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2022 | | | | | |  | Проверил  Преподаватель кафедры МОВС  Пономарев Ф.А.  (подпись)  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2022 | | | |  |
|  | | | | | | | | | | | | |
|  | | | Пермь 2022 | | | | | | |  | | |

СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc97220105)

[1 Постановка задачи 3](#_Toc97220106)

[2 Структура компилятора 5](#_Toc97220107)

[3 Описание языка Pascal 8](#_Toc97220108)

[4 Модуль ввода-вывода 9](#_Toc97220109)

[4.1 Модуль ввода 9](#_Toc97220110)

[4.1.1 Проектирование 9](#_Toc97220111)

[4.1.2 Тестирование 9](#_Toc97220112)

[4.2 Модуль вывода 9](#_Toc97220113)

[5 Лексический анализатор 10](#_Toc97220114)

[6 Синтаксический анализатор 11](#_Toc97220115)

[7 Семантический анализатор 12](#_Toc97220116)

[8 Генерация кода 13](#_Toc97220117)

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).

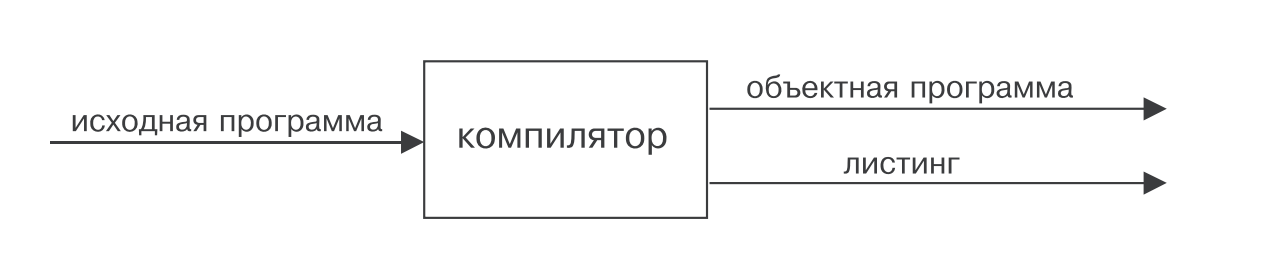
*Индивидуальная часть.*

Ссылочные типы данных. Описание функций. Вызов функции и указатели в выражении.

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.

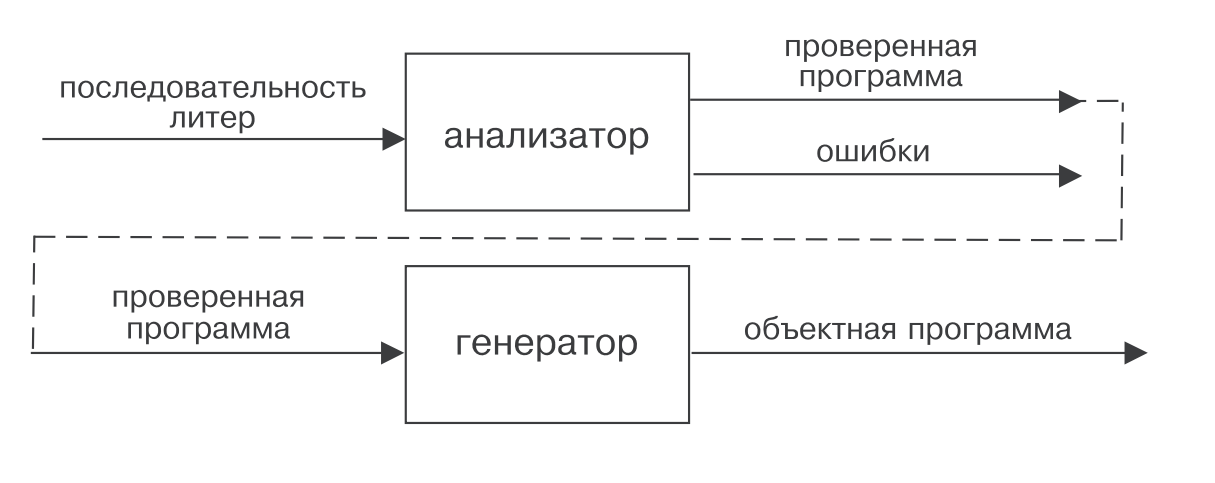
Компилятор можно представить в виде следующей схемы:



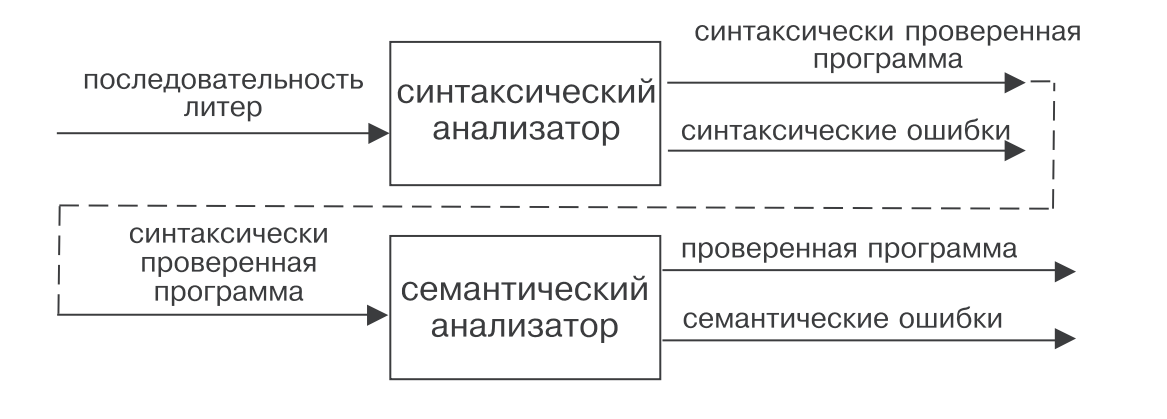
Работу компилятора можно разделить на 2 этапа:

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:



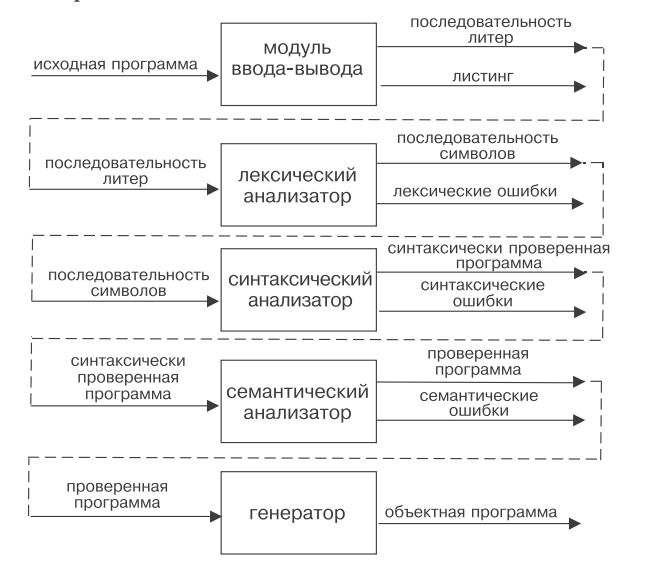
Синтаксический анализатор проверяет, удовлетворяет ли программа формальным правилам. Назначение же семантического анализатора состоит в том, чтобы выяснить, не нарушены ли неформальные правила описания языка.

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:

|  |  |
| --- | --- |
| Лексема | Описание |
| program | Название программы |
| begin |  |
| end |  |
| var |  |
| type |  |
| function |  |
| if |  |
| then |  |
| else |  |
| while |  |
| do |  |
| and |  |
| or |  |
| xor |  |
| not |  |
| @ | Оператор адреса |
| ^ | Оператор переадресации указателя |
| + | Сложение |
| - | Вычитание |
| \* | Умножение |
| / | Деление |
| ( | Открывающая скобка |
| ) | Закрывающая скобка |
| { | Начало комментария |
| } | Конец комментария |
| := | Оператор присваивания |
| . |  |
| , |  |
| : |  |
| ; |  |
| = |  |
| < |  |
| > |  |
| <= |  |
| >= |  |
| <> |  |
| (\* |  |
| \*) |  |
|  |  |
|  |  |

4 Модуль ввода-вывода

4.1 Модуль ввода

4.1.1 Проектирование

Для реализации модуля ввода создадим класс Reader, который будет содержать следующие поля и методы:

class Reader {

public:

Reader(std::istream& stream);

~Reader();

std::pair<char, TextPosition> get();

private:

std::istream& stream;

TextPosition pos;

};

stream хранит в себе ссылку на поток ввода, таким образом в качестве ввода могут использоваться различные источники (из файла, из строки, из консоли и т.д.).

pos хранит в себе информацию о текущей позиции в тексте и имеет следующее описание:

struct TextPosition {

int lineIndex;

int charIndex;

TextPosition();

TextPosition(int lineIndex, int charIndex);

};

То есть он хранит номер текущей строки и номер текущей позиции в этой строке.

Метод get() возвращает следующий символ из потока ввода и его позицию

4.1.2 Тестирование

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

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Видим, что в консоль программа вывелась такая же, как и была на входе.

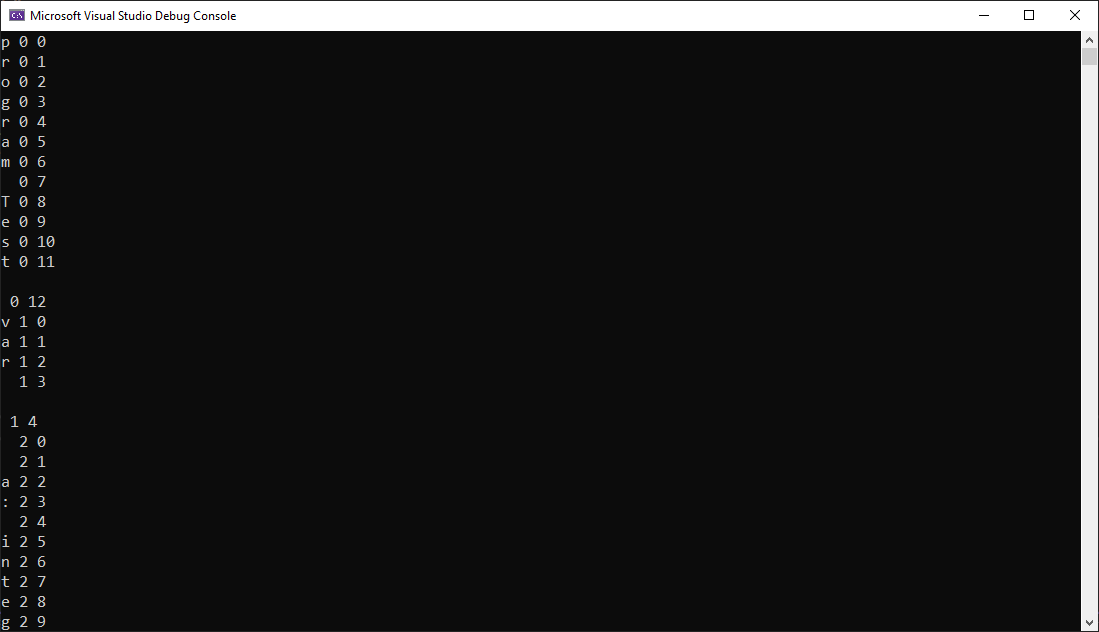
Теперь протестируем поток ввода из строки, для этого запишем следующую программу:

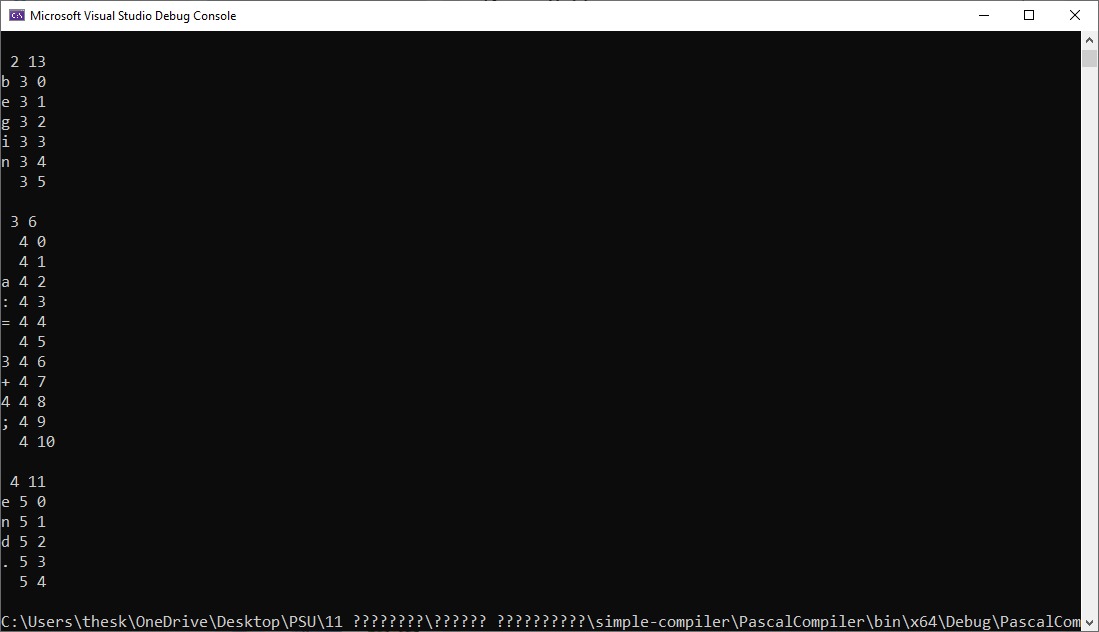
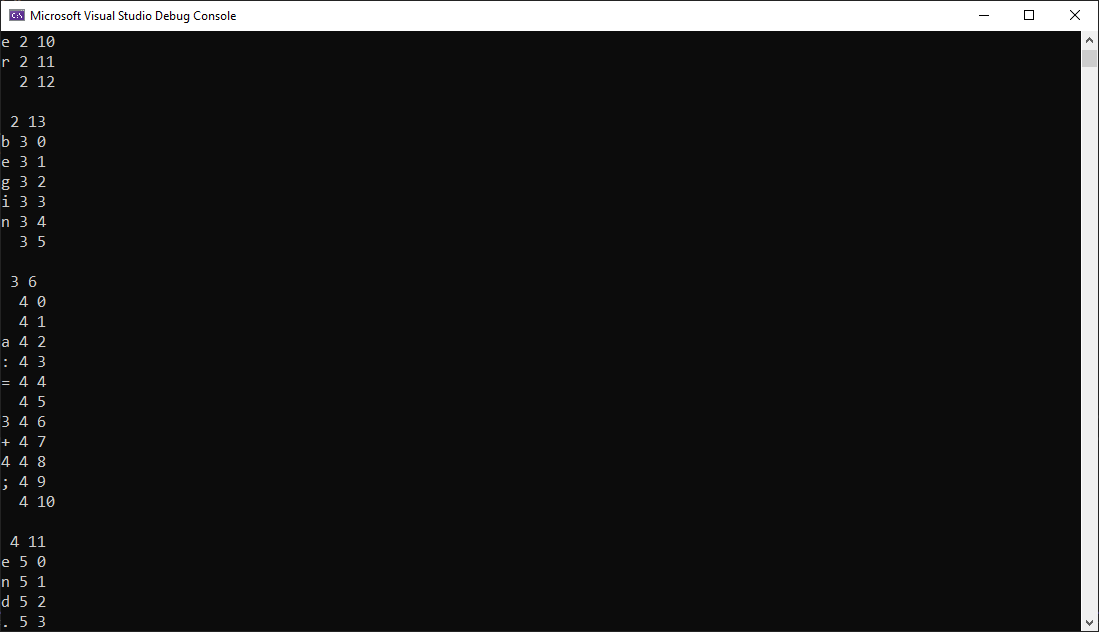
std::string s = "program Test\nvar \n a: integer \nbegin \n a:= 3+4; \nend.";

Результат:

Изображение выглядит как текст

Автоматически созданное описание

Теперь будем выводить позиции символов, получим следующий результат:



4.2 Модуль вывода

5 Лексический анализатор

Лексический анализатор формирует токены исходной программы, кроме того, он распознает и исключает комментарии, которые не нужны для дальнейшей трансляции.

В первую очередь необходимо описать токены. Для этого создадим абстрактный класс CToken, который имеет следующую сигнатуру:

class CToken {

public:

CToken(TokenType tType);

TokenType getType();

virtual std::string toString() = 0;

private:

TokenType tType;

};

Поле tType хранит в себе тип токена (идентификатор, ключевое слово или константа), метод getType() возвращает тип токена.

Далее опишем классы для токена идентификатора, ключевого слова и константы, каждый из которых будет производным от класса CToken.

class CIdentToken : public CToken {

public:

CIdentToken(std::string name);

std::string toString() override;

private:

std::string name;

};

class CKeywordToken : public CToken {

public:

CKeywordToken(KeyWords name);

std::string toString() override;

private:

KeyWords name;

};

class CConstToken : public CToken {

public:

CConstToken(CVariant\* value);

std::string toString() override;

private:

std::unique\_ptr<CVariant> value;

};

Класс CConstToken содержит в себе поле типа CVariant – это класс, который описывает какого типа константа (integer, real, string, boolean).

//Надо наверно описать тут еще keywords и прочее

6 Синтаксический анализатор

Здесь описание синтаксического анализатора.

7 Семантический анализатор

Здесь описание семантического анализатора.

8 Генерация кода

Здесь генерация кода.