# Синтаксический и семантический анализатор

## Постановка задачи

Нужно написать синтаксический и семантический анализатор языка Паскаль.

Синтаксический анализатор должен пройтись по набору токенов и проверить правильность грамматик и синтаксических конструкций языка. Если при проверке грамматики обнаружено несоответствие (синтаксическая ошибка), то программа должна выполнить нейтрализацию этой ошибки (пропустить несколько токенов до необходимой позиции чтобы продолжить компиляцию).

Семантический анализатор должен хранить информацию о всех объявленных идентификаторах (переменных, типах, константах), а также проверять исполнение неформальных соглашений языка Паскаль (соответствие типов данных в выражении). Если какие-то неформальные соглашения не выполнены (семантическая ошибка), то должна быть выполнена нейтрализация этой ошибки, чтобы продолжить компиляцию

## Проектирование

### Синтаксический анализатор

В начале нужно определиться с грамматиками. Они представлены в виде БНФ в отдельном файле проекта «БНФ Паскаль.txt»

Дальше было принято решение компиляцию каждой отдельной грамматики записать в виде функции вида: compile\_<Имя грамматики>(): bool. И того получилось 24 функции, распознающих 24 грамматики. Все функции представлены на рисунке 2.1.

bool compile\_Program ();

bool compile\_Block ();

bool compile\_VarSection ();

bool compile\_VarDeclaration ();

bool compile\_Type ();

bool compile\_OperatorSection ();

bool compile\_CompoundOperator ();

bool compile\_Operator ();

bool compile\_IfOperator ();

bool compile\_WhileOperator ();

bool compile\_Variable ();

bool compile\_Assignment ();

bool compile\_Expression ();

bool compile\_SimpleExpression ();

bool compile\_Term (); //слагаемое

bool compile\_Multiplier (); //множитель

//Индивидуальное задание

bool compile\_TypeSection ();

bool compile\_TypeDeclatation ();

bool compile\_Record ();

bool compile\_WithOperator ();

bool compile\_ForOperator ();

bool compile\_RepeatOperator ();

Рисунок 2.1 – список всех функций - грамматик

Каждая функция возвращает Boolean метку об успешности или не успешности компиляции.

Все функции объявлены как методы класса – Компилятора, поэтому выполнение каждой функции влечет за собой изменение состояния объекта – Компилятора и потому эти функции нельзя назвать чистыми.

Класс Компилятор – общий класс, выполняющий все функции компиляции:

* Синтаксический анализ
* Семантический анализ
* Обработка ошибок
* Хранение областей видимости (для семантического анализа)

Описание класса представлено на рисунке 2.2.

class Compiler

{

public:

Compiler();

~Compiler();

void bindReader(FileReader\* reader);

ErrorsArray Compile(void\* const program);

private:

bool readToken();

AbstactToken\* last\_token; //только что прочитанный токен

AbstactToken\* next\_token; //следующий (еще не прочитанный) токен

comp\_error\_t next\_error; //ошибка следующего (еще не прочитанного) токена

LexicalProcessor lexer; //лексический анализатор

Scope \*global\_scope; //глобальная область видимости

AbstractType\* last\_compiled\_type; //последний скомпилированный тип

Data\* last\_compiled\_variable; //последняя скомпилированная переменная

//условия токенов

bool isNextTokenCorrect(); //следующий токен - корректный/правильный

bool isNextTokenOperator(); //следующий токен - начало оператора

bool isNextTokenExpression(); //следующий токен - начало выражения

bool isNextTokenType(); //следующий токен - начало типа данных

bool isNextTokenCompareOperator(); //следующий токен - оператор сравнения

bool isNextTokenVariable(); //следующий токен - переменная

bool isAssignableTypes(AbstractType\* t1, AbstractType\* t2); //типы соотносимы

// Грамматики

…

}

Рисунок 2.2 – Описание класса Компилятор

### Семантический анализатор

Семантический анализатор должен хранить информацию о типах и идентификаторах в областях видимости. Были созданы специальные структуры – Данные программы (оболочка над указателем на переменную программы с указанием типа данных), Типы и Область видимости. Диаграмма классов представлена на рисунке 2.3.

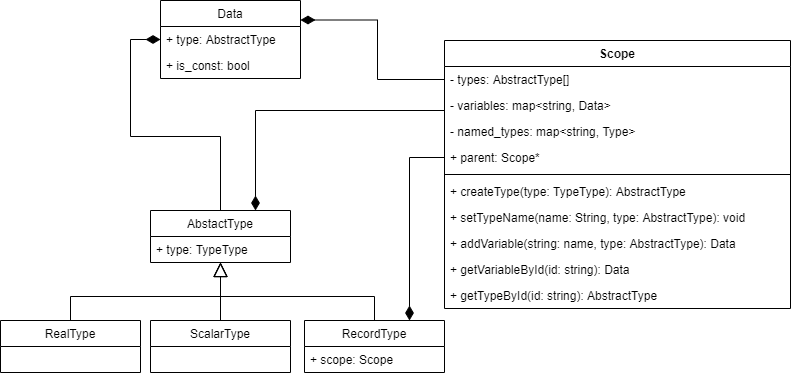


Рисунок 2.3 – Диаграмма классов для областей видимости

## Реализация

## Тестирование

Было придумано два теста: программа с множеством ошибок и правильная программа.

На рисунке 4.1 представлен тест неправильной программы. А на рисунке 4.2 представлен вывод ошибок компилятора.

program test;

type T = record x, y, z: Real; a, b, c: Integer; end;

var a, b, c: Integer; d, e: Real; x: T;

begin

a := 10; b := -10.01; c := 'test123'; d:= (0.5 + x) / (PI \* 180 / 360) \* 2;

d[5] := +0.0; d[3] := -1; y := 2 \* f - 1.5;

if (a > b) and (b <> c) then begin x1 := map + 1.1

a :=

b := c

end

x.x = 0.0;

while (a > b) do begin x := 3; y := x - := 5;end

with x do x := 5

d := 100000000000000000000000;

for i := 0 to 5 do begin end;

repeat begin end until a < b;

end

Рисунок 4.1 – Неправильная программа

ERROR 24(13, 15): not assignable types

ERROR 24(14, 13): not assignable types

ERROR 23(14, 18): unknown identifier (variable or function or procedure)

ERROR 12(14, 18): expected ')'

ERROR 11(15, 2): expected ':='

ERROR 11(16, 2): expected ':='

ERROR 23(16, 12): unknown identifier (variable or function or procedure)

ERROR 23(17, 10): unknown identifier (variable or function or procedure)

ERROR 24(18, 25): not assignable types

ERROR 23(19, 6): unknown identifier (variable or function or procedure)

ERROR 23(20, 8): unknown identifier (variable or function or procedure)

ERROR 3(22, 8): expected ';'

ERROR 24(27, 8): not assignable types

ERROR 23(27, 9): unknown identifier (variable or function or procedure)

ERROR 18(28, 11): expected identifier or constant or '('

ERROR 1(33, 30): too long number for variable

ERROR 9(44, 4): expected '.'

Рисунок 4.2 – Вывод ошибок компилятора

На рисунке 4.3 представлена правильная программа. Вывода быть не должно, потому что компилятор выводит только сообщения об ошибках.

program test;

type T = record

x,y,z: Real;

end;

var a,b,c: Integer;

x,y: Real;

t: T;

begin

a := 5; b := -10.01; c := +0; x := 3;

t.x := x; t.y := y; t.z := 0.0;

for a := 1 to 5 do

begin b := b + 1; end;

with t do x := 3;

if (a > b) then

begin a := 0; end

else

begin b := 0; end;

end.

Рисунок 4.3 – Правильная программа

Компилятор ничего не выводит