

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФГБОУ ВПО «Пермский государственный национальный исследовательский
университет»

ОТЧЕТ

по дисциплине «Формальные грамматики и методы
трансляции»

Работу выполнила
студентка гр. ПМИ-1,2
Калинина М.О. _____
(подпись)
«___» _____ 2022

Проверил
ассистент кафедры МОВС
Пономарёв Ф.А.

(подпись)
«___» _____ 2022

Пермь 2022

Оглавление

Модуль ввода-вывода	3
Описание	3
Проектирование	3
Реализация	3
Тестирование	9
Лексический анализатор.....	12
Описание	12
Проектирование	12
Реализация программы.....	13
Тестирование	17
Синтаксический анализатор.....	21
Описание	21
Проектирование	21
Реализация	23
Тестирование	35
Семантический анализатор	40
Описание	40
Проектирование	40
Реализация	41
Тестирование	48

Модуль ввода-вывода

Описание

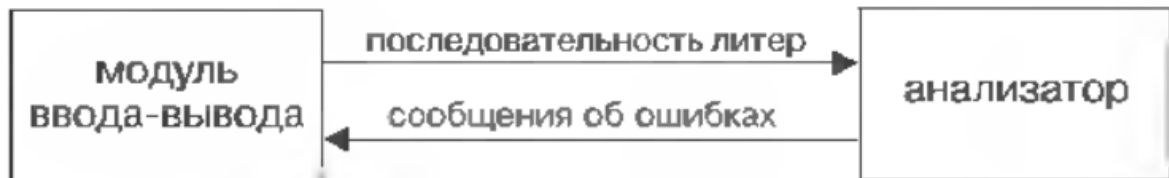
Модуль ввода-вывода считывает последовательность литер исходной программы с внешнего устройства и передает их анализатору.

Проектирование

На данном этапе нам необходимо входной файл разбить на лексемы, в соответствии с синтаксисом языка, а именно его ключевыми словами и специальными символами, определив для каждой лексемы её тип и сохраняя их в определённый буфер.

Концепция данного модуля заключается на обработке отдельных групп символов. Набор всех допустимых символов, был разбит на 3 подмножества.

- 1) Множество букв в нижнем и верхнем регистре и нижнее подчёркивание
- 2) Множество цифр
- 3) Множество символов



Реализация

В первую очередь модуль ввода-вывода распознает операторы, которые могут быть разделителями.

```

internal string GetLetter()
{
    string str = "";
    while (curLine != null) {
        switch (curLetter)
        {
            case ';':
            case ')':
            case '{':
            case '}':
            case '[':
            case ']':
            case '(':
            case ',':
            case '/':
            case '*':
            case '+':
            case '-':
            case '=':

                if (str != "")

                    return str;

            str = "" + curLetter;
            GetNext();
            return str;
        }
    }
}

```

Всё что находится в кавычках он определяет, как одну лексему.

```

case '\n':

    if (str != "")

        return str;

    str = "" + curLetter; GetNext();
    while (curLine != null && curLetter != '\n')
    {
        str += curLetter; GetNext();
    }
    if (curLine != null ) { str += curLetter; GetNext(); return str; }

    break;
case '\':

    if (str != "")

        return str;

    str = "" + curLetter; GetNext();
    while (curLine != null && curLetter != '\')
    {
        str += curLetter; GetNext();
    }
    if (curLine != null ) { str += curLetter; GetNext(); return str; }

```

Комментарии он просто пропускает.

```
case '{':
    if (str != "")

        return str;

    str = "" + curLetter; GetNext();
    while (curLine != null && curLetter != '}')
    {
        str += curLetter; GetNext();
    }
    if (curLine != null) { str += curLetter; GetNext(); }
    break;
```

Также реализована проверка символов наперед для неодносимвольных операторов.

```

        break;
    case ':':

        if (str != "")

            return str;

        GetNext();
        if (curLine != null && curLetter == '=')
        {
            GetNext();
            return ":@";
        }

        else
            return ":";

    case '<':
        if (str != "")    return str;
        GetNext();
        if (curLine != null && curLetter == '=')
        {
            GetNext();
            return "<=";
        }

        else if (curLine != null && curLetter == '>')
        {
            GetNext();
            return "<>";
        }
        else
            return "<";

```

```

    case '>':
        if (str != "")

            return str;

        GetNext();
        if (curLine != null && curLetter == '=')
        {
            GetNext();
            return ">=";
        }

        else
            return ">";

```

При встрече символов переноса строки, пробела, табуляции модуль ввода-вывода их просто пропускает. Если встретился другой символ, он прибавляет его в строку пока не встретился известный нам оператор.

```
case ' ':
case '\\0':
case '\\t':
case '\\n':
    if (str != "")
    {
        ...    GetNext(); return str;
    }
    GetNext();

    break;
default:
    str += curLetter;
    GetNext();
    break;
```

На модуль ввода-вывода положен также вывод ошибок каждой строки программы.

```

public void GetNext()
{
    if ( curPosition == curLine.Length - 1)
    {
        outputFile.WriteLine(curLine);
        curLine = inputFile.ReadLine();
        curPosition = -1;

        if (errorsInLine != null)
        {
            foreach (Error e in errorsInLine)
                if (e.position >= 0)
                    e.InputError(outputFile);
            errorsInLine.Clear();
        }
        curLetter = '\0';
    }
    else
    {
        curPosition++;
        curLetter = curLine[curPosition];
    }
    if (curLine == null)
    {
        inputFile.Close();
        outputFile.Close();
    }
}

```


Тестирование

Для данной программы

```
var sr,sf,ss:string;
{комментарий}
function BinAdd(s1,s2:string):string;
var s:string; l,i,d,carry:byte;
begin

    if length(s1)>length(s2) then while length(s2)<length(s1) do s2:='0'+s2
                                else while length(s1)<length(s2) do s1:='0'+s1;
    l:=length(s1);
    s:=''; carry:=0;
    for i:=l downto 1 do begin
        d := (ord(s1[i])-ord('0')) + (ord(s2[i])-ord('0')) + carry;
        carry := d div 2;
        d:=d mod 2;
        s:=char(d+ord('0')) + s;
    end;
    if carry<>0 then s:='1'+s;
    BinAdd:=s;
end;

begin
    writeln("введите 1-е двоичное число:");
    readln(sf);
    writeln("введите 2-е двоичное число:");
    readln(ss);
    sr:=BinAdd(sf,ss);
    writeln("результат сложения = ",sr);
end.
```

был выведен данный результат:

```

var
sr
,
sf
,
ss
:
string
;
function
BinAdd
(
s1
,
s2
:
string
)
:
string
;
var
s
:
string
;
l
,
i
,
d
,
carry
:
byte
;
begin
if
length
(
s1
)
>
length
(
s2
)
then
while

```

Был вставлен неполный скриншот, но из этой части уже можно понять, что модуль работает верно.

Лексический анализатор

Описание

Лексический анализатор формирует символы исходной программы и строит их внутреннее представление. Кроме того, сканер распознает и исключает комментарии, которые не нужны для дальнейшей трансляции.

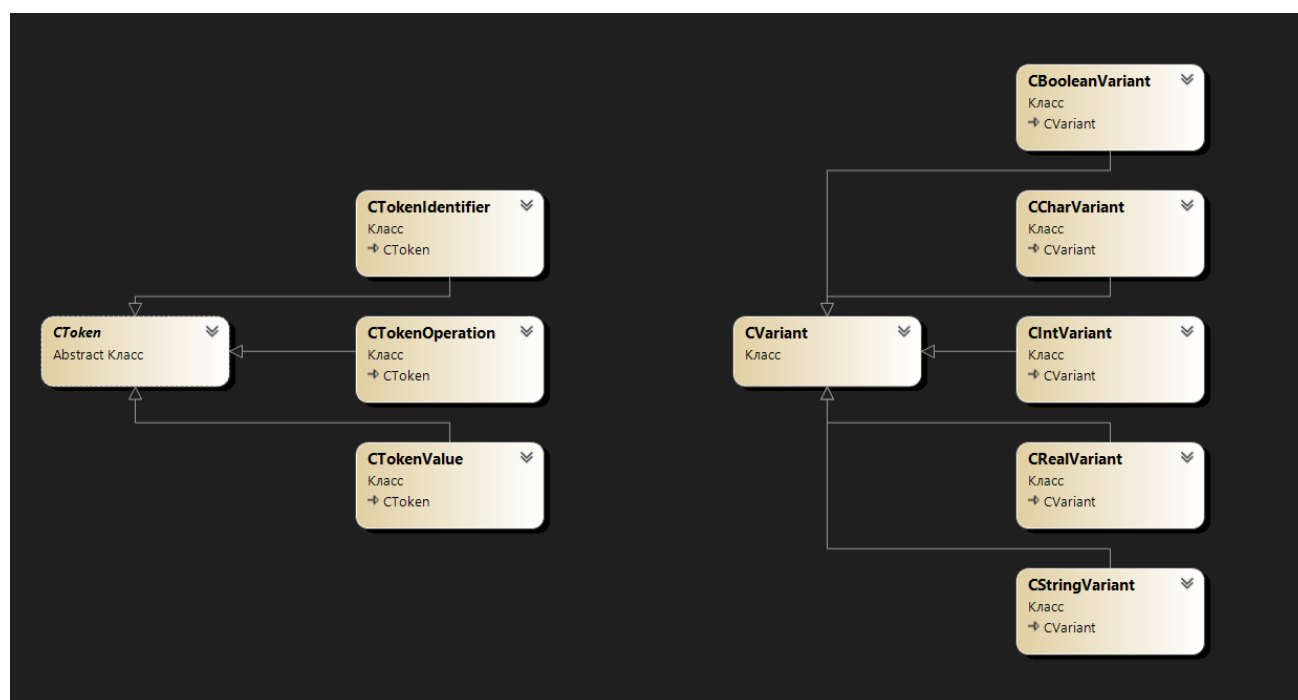
Проектирование

Нам необходимо понять, как создать структуру классов и ошибок.

Для каждой группы символов нужно будет создать свой отдельный класс, который будет связан иерархически.

Остановимся на данной реализации, где есть класс родитель для всех остальных типов (идентификатора, спец символа, константы).

И для символа констант сделаем такой же родительский класс для типов констант (целого, логического, вещественного строкового)



Реализация программы

Все спец слова должны хранится в специальном словаре, где ключ — это строка слова, а значение - его код, базовые типы были отнесены туда же.

Так же был создан класс для хранения ошибок, а внутри него словарь для ошибок и их описаний.

```

class Error
{
    static public Dictionary<int, string> errors = new Dictionary<int, string>
    {
        {1,"Отсутствует знак операции"},
        {2,"должно идти имя"},
        {3,"должен идти операнд"},
        {4,"знак операции отсутствует или неверного типа"},
        {5,"в исходном тексте найден неверный символ"},
        {6,"Ошибка в вещественной константе"},
        {7,"Ошибка в целой константе"},
        {8,"должно идти слово" },
        {9,"недопустимый тип"},
        {10,"Повторное описание переменной"},
        {11,"Строка слишком длинная"},
        {12,"Неописанная переменная"},
        {13,"Ожидалось выражение логического типа"},
        {14,"Несоответствие типов"},
        {15,"Неверный тип операндов"}
    };

    public int position;
    private int code;
    private string word;
    public Error(int position, int code)
    {
        this.position = position;
        this.code = code;
    }
    public Error(int position, int code, string word)
    {
        this.position = position;
        this.code = code;
        this.word = word;
    }
    public void InputError(StreamWriter outputFile)
    {
        outputFile.WriteLine("Ошибка! Код: " + code);
        outputFile.WriteLine(errors[code] + " " + word);
        outputFile.WriteLine();
    }
}

```

Ошибки хранились в листе ошибок и выводились после каждой строки программы.

Ошибки, связанные с преобразованием типа, обрабатывались непосредственно в классе, связанном с ЭТИМ ТИПОМ.

Классификация лексем на символы

Распознавание ключевых слов или идентификаторов

```
if (keyWords.TryGetValue(s, out value))
{
    curToken = new CTokenOperation(TokenType.Operation, keyWords[s]); return curToken;
}
else
{
    if (s[0] >= 'A' && s[0] <= 'Z' || s[0] >= 'a' && s[0] <= 'z')
    {
        string name = "";
        int i = 0;
        while (i < s.Length && (s[i] >= 'A' && s[i] <= 'Z' || s[i] >= 'a' && s[i] <= 'z' || s[i] >= '0' && s[i] <= '9'))
        {
            name += s[i].ToString();
            ++i;
        }
        name = name.ToLower();
        if (name == "false" || name == "true")
        {
            curToken = new CTokenValue(TokenType.Value, value: new CBooleanVariant(name));
            return curToken;
        }
        else if (keyWords.ContainsKey(name))
            curToken = new CTokenOperation(TokenType.Operation, keyWords[name]);

        else
            curToken = new CTokenIdentifier(TokenType.Identifier, name);
        return curToken;
    }
}
```

Распознавание вещественных и целых чисел

```

else if (s[0] >= '0' && s[0] <= '9')
{
    string number = "";
    int i = 0;
    while (i < s.Length && s[i] >= '0' && s[i] <= '9')
    {
        number += s[i].ToString();
        ++i;
    }
    if (i < s.Length && s[i] == '.')
    {
        number += s[i].ToString();
        ++i;
        while (i < s.Length)
        {
            if (s[i] >= '0' && s[i] <= '9')
                number += s[i].ToString();
            else IO.errorsInLine.Add(item: new Error(position: IO.curPosition + 1, code: 5));
            ++i;
        }
        number = number.Replace(oldChar: '.', newChar: ',');
        curToken = new CTokenValue(tt: TokenType.Value, value: new CRealVariant(number, IO));
    }
    else
        curToken = new CTokenValue(tt: TokenType.Value, value: new CIntVariant(number, IO));
}
}

```

Распознавание стрингов и чаров

```

else if (s[0] == '\\' && s.Length <= 3 && s[s.Length-1] == '\\')

    curToken = new CTokenValue(tt: TokenType.Value, value: new CCharVariant(s));
else if (s[0] == '\"' && s[s.Length - 1] == '\"')

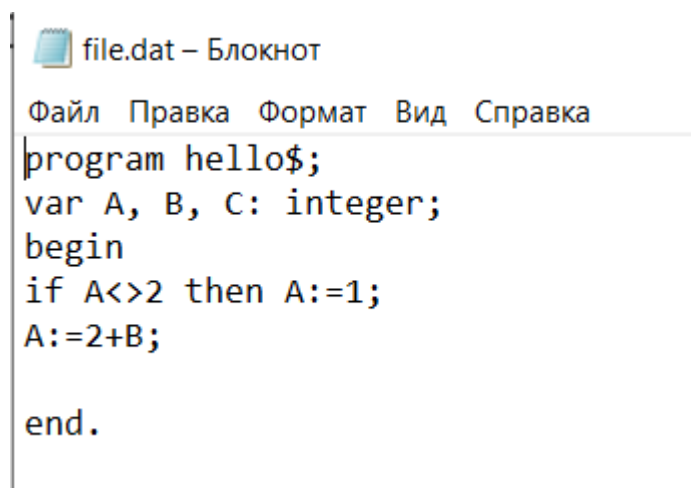
    curToken = new CTokenValue(tt: TokenType.Value, value: new CStringVariant(s, IO));

```


Тестирование

1) Входные данные:

Имеется ошибка, символ не входящий в область допустимых символов

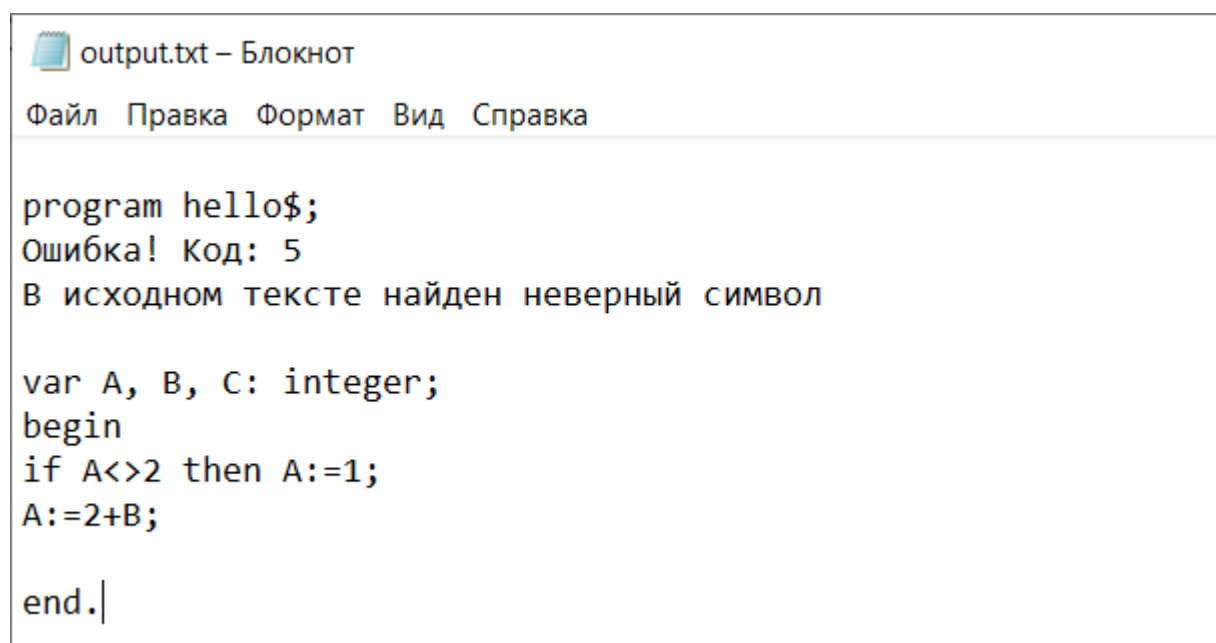


```
file.dat – Блокнот
Файл  Правка  Формат  Вид  Справка
program hello$;
var A, B, C: integer;
begin
if A<>2 then A:=1;
A:=2+B;

end.
```

Ожидаемый результат: сообщение об ошибке

Выходной результат:



```
output.txt – Блокнот
Файл  Правка  Формат  Вид  Справка

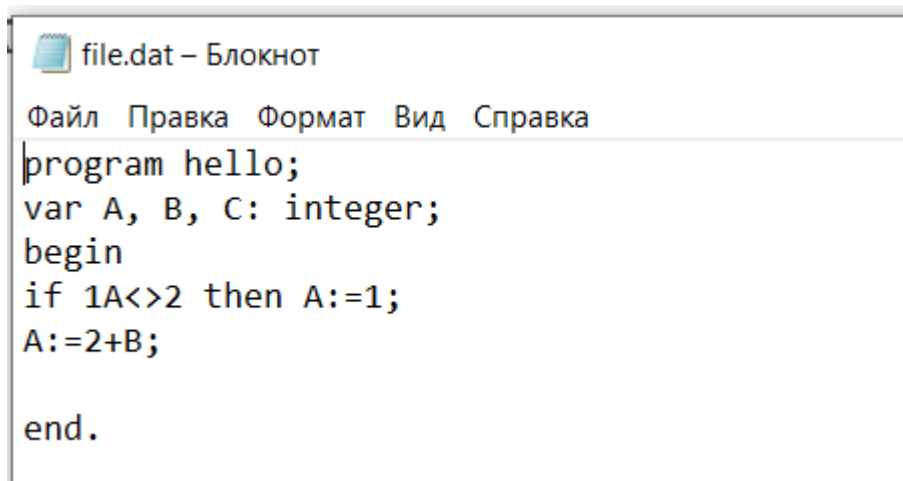
program hello$;
Ошибка! Код: 5
В исходном тексте найден неверный символ

var A, B, C: integer;
begin
if A<>2 then A:=1;
A:=2+B;

end.
```

2) Исходный файл:

Имеется ошибка, символ не входящий в область допустимых символов

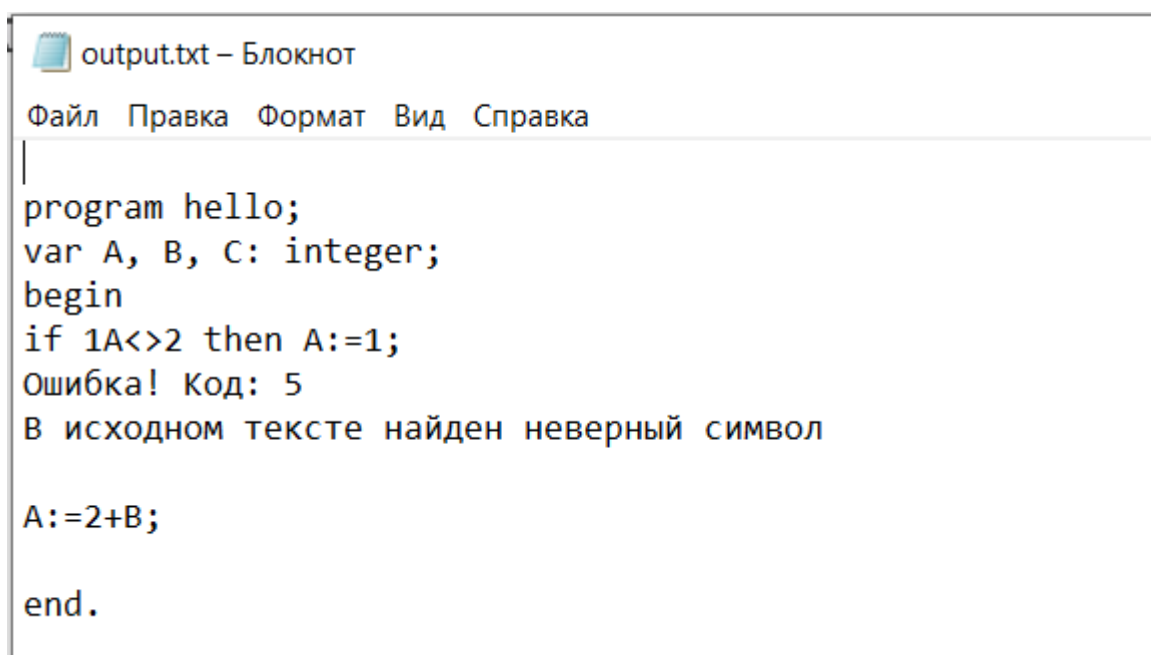


```
file.dat – Блокнот
Файл  Правка  Формат  Вид  Справка
program hello;
var A, B, C: integer;
begin
if 1A<>2 then A:=1;
A:=2+B;

end.
```

Ожидаемый результат: сообщение об ошибке

Выходной результат:



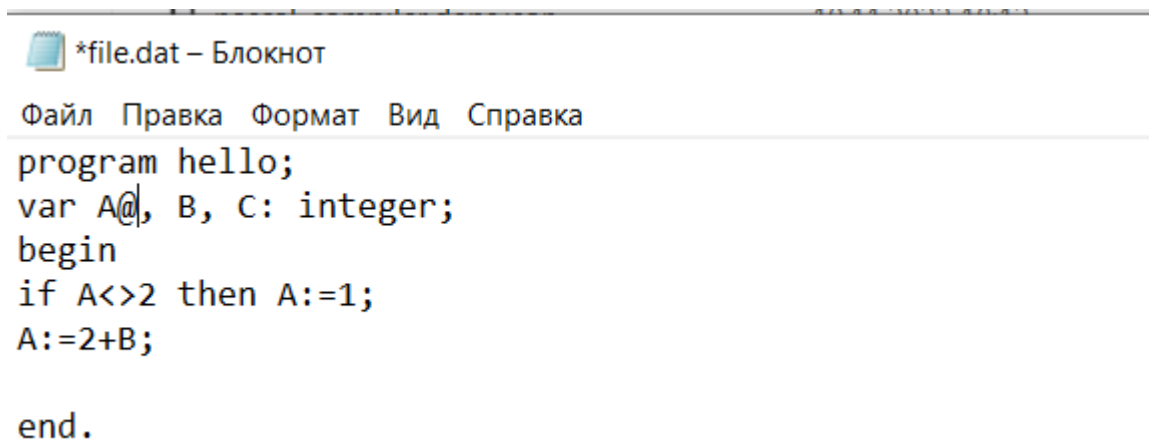
```
output.txt – Блокнот
Файл  Правка  Формат  Вид  Справка
|
program hello;
var A, B, C: integer;
begin
if 1A<>2 then A:=1;
Ошибка! Код: 5
В исходном тексте найден неверный символ

A:=2+B;

end.
```

3) Исходный файл

Имеется ошибка, символ не входящий в область допустимых символов

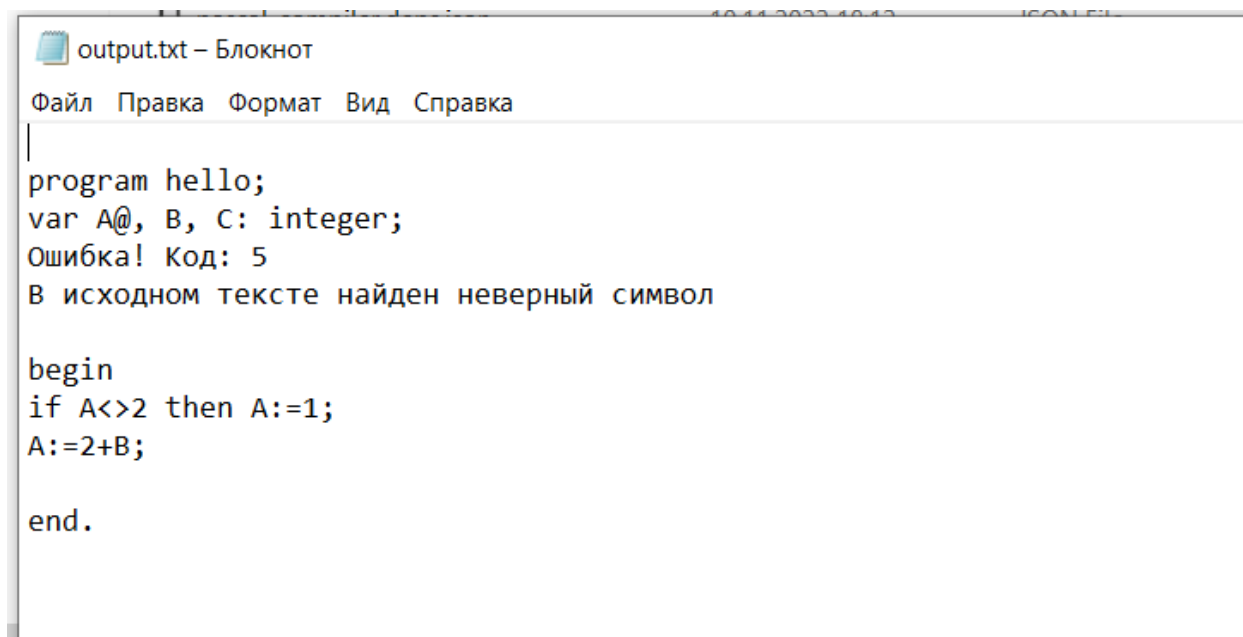


```
*file.dat – Блокнот
Файл  Правка  Формат  Вид  Справка
program hello;
var A@, B, C: integer;
begin
if A<>2 then A:=1;
A:=2+B;

end.
```

Ожидаемый результат: сообщение об ошибке

Выходной результат:




```
output.txt – Блокнот
Файл  Правка  Формат  Вид  Справка
|
program hello;
var A@, B, C: integer;
Ошибка! Код: 5
В исходном тексте найден неверный символ

begin
if A<>2 then A:=1;
A:=2+B;

end.
```

4) Исходный файл

Ошибок нет

 file.dat – Блокнот


Файл Правка Формат Вид Справка

```
program hello; {комментарий}
var A, B, C: integer;
begin
if A<>2 then A:=1;
A:=2+B;

end.
```

Ожидаемый результат: нет сообщений об ошибке

Выходной результат:

 output.txt – Блокнот

Файл Правка Формат Вид Справка

```
program hello; {комментарий}
var A, B, C: integer;
begin
if A<>2 then A:=1;
A:=2+B;
end.
```

Синтаксический анализатор

Описание

Синтаксический анализатор проверяет правильность и порядок использования символов языка.

Проектирование

Синтаксический анализатор строится на правилах использования символов языка, в нашем случае используются БНФ.

Ниже приведены все использованные БНФ:

$\langle \text{программа} \rangle ::= \text{program } \langle \text{имя} \rangle; \langle \text{блок} \rangle.$

$\langle \text{имя} \rangle ::= \langle \text{буква} \rangle \{ \langle \text{буква} \rangle | \langle \text{цифра} \rangle \}$

$\langle \text{блок} \rangle ::= \langle \text{раздел констант} \rangle \langle \text{раздел переменных} \rangle \langle \text{оператор} \rangle$

$\langle \text{раздел констант} \rangle ::= \langle \text{пусто} \rangle | \text{const } \langle \text{определение константы} \rangle; \{ \langle \text{определение константы} \rangle; \}$

$\langle \text{определение константы} \rangle ::= \langle \text{имя} \rangle = \langle \text{константа} \rangle$

$\langle \text{константа} \rangle ::= \langle \text{число без знака} \rangle | \langle \text{знак} \rangle \langle \text{число без знака} \rangle |$

$\langle \text{имя константы} \rangle | \langle \text{знак} \rangle \langle \text{имя константы} \rangle | \langle \text{строка} \rangle$

$\langle \text{число без знака} \rangle ::= \langle \text{целое без знака} \rangle | \langle \text{вещественное без знака} \rangle$

$\langle \text{целое без знака} \rangle ::= \langle \text{цифра} \rangle \{ \langle \text{цифра} \rangle \}$

$\langle \text{вещественное без знака} \rangle ::= \langle \text{целое без знака} \rangle . \langle \text{цифра} \rangle \{ \langle \text{цифра} \rangle \} | \langle \text{целое без знака} \rangle . \langle \text{цифра} \rangle \{ \langle \text{цифра} \rangle \} \text{E} \langle \text{порядок} \rangle | \langle \text{целое без знака} \rangle \text{E} \langle \text{порядок} \rangle$

$\langle \text{порядок} \rangle ::= \langle \text{целое без знака} \rangle | \langle \text{знак} \rangle \langle \text{целое без знака} \rangle$

$\langle \text{знак} \rangle ::= + | -$

<имя константы>::=<имя>

<строка>::='<символ>{<символ>}'

<раздел переменных>::= var <описание однотипных переменных>;{<описание однотипных переменных>;}<пусто>

<описание однотипных переменных>::=<имя>{,<имя>}:<тип>

<оператор>::=<простой оператор>|<сложный оператор>

<простой оператор>::=<оператор присваивания>|<пустой оператор>

<оператор присваивания>::=<имя>:=<выражение>

<выражение>::=<простое выражение>|<простое выражение><операция отношения><простое выражение>

<операция отношения>::=|<>|<|<=>|=|>|in

<простое выражение>::=<знак><слагаемое>{<аддитивная операция><слагаемое>}

<аддитивная операция>::=+|-|or

<слагаемое>::=<множитель>{<мультипликативная операция><множитель>}

<мультипликативная операция>::=*/|div|mod|and

<множитель>::=<имя>|<константа без знака>|(<выражение>)

<константа без знака>::=<число без знака>|<строка>|<имя константы>|nil

<пустой оператор>::=<пусто>

<пусто>::=

<сложный оператор>::=<составной оператор>|<выбирающий оператор>|<оператор цикла>

<составной оператор> ::= begin <оператор> {;<оператор>} end

<выбирающий оператор> ::= if <выражение> then <оператор> | if <выражение> then <оператор> else <оператор>

<оператор цикла> ::= while <выражение> do <оператор>

Для обработки событий, что наша программа не соответствует БНФ будем использовать лист ошибок и выводить ошибки после каждой строки программы.

Принятие решений при выборе альтернатив положим на этом же уровне БНФ, анализируя текущий символ.

Так как синтаксический анализатор тесно взаимодействует с семантическим, мы будем рассматривать в этой части отчёта части семантического анализатора.

Реализация

Проверка соответствия спец символа

```
public bool Accept(TokenType type, string keyWord)
{
    if (!(curToken is CTokenOperation) || ((CTokenOperation)curToken).code != LexicalAnalyzer.keyWords[keyWord])
        return false;

    return true;
}
```

Все методы запрограммированы в соответствии БНФ

```

void Block() // <блок>::=<раздел констант><раздел переменных><оператор>
{
    try
    {
        ConstArea();
    }
    catch
    {
    }

    try
    {
        VarArea();
    }
    catch
    {
        la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 9));
    }

    try
    {
        Operator();
    }
    catch
    {
    }
}

```

```

void ConstArea() // <раздел констант>::=<нусто>|const <определение константы>;{<определение константы>;}
{
    if (Accept(TokenType.Operation, "const"))
    {
        ConstDeter();
        if (!Accept(TokenType.Operation, ";"))
            la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 3));
        GetNextToken();
        while (true)
        {
            try
            {
                ConstDeter();
            }
            catch
            {
                break;
            }
            if (!Accept(TokenType.Operation, ";"))
                la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 3));
            GetNextToken();
        }
    }
}

```



```

void ConstDeter() // <определение константы>::=<имя>=<константа>
{
    if (curToken.tt != TokenType.Identifier)
    {
        throw new Exception();
    }
    GetNextToken();
    if (!Accept(TokenType.Operation, "="))
    {
        la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 3));
    }
    GetNextToken();
    Const();
}

```

```

void Const() // <константа>::=<число без знака>|<знак><число без знака>|
            //<имя константы>|<знак><имя константы>|<строка>
            //<число без знака>::=<целое без знака>|<вещественное без знака>
            //<целое без знака>::=<цифра>{<цифра>}
            //<вещественное без знака>::=<целое без знака>.<цифра>{<цифра>}|<ц
            //<порядок>::=<целое без знака>|<знак><целое без знака>
            //<знак>::=+|-
            //<имя константы>::=<имя>
            //<строка>::='<символ>{<символ>}'

{
    if (Accept(TokenType.Operation, "-"))
    {
        GetNextToken();
        if (curToken.tt == TokenType.Value || curToken.tt == TokenType.Identifier)
        {
            GetNextToken();
        }
        else
        {
            la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 9));
        }
    }
    else
    {
        if (curToken.tt == TokenType.Value || curToken.tt == TokenType.Identifier)
        {
            GetNextToken();
        }
        else
        {
            la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 9));
        }
    }
}
}

```

```

void VarArea() //<раздел переменных>:= var <описание однотипных переменных>{<описание однотипных переменных>}|<пусто>
{
    if (Accept(TokenType.Operation, "var"))
    {
        GetNextToken();
        Var();
        if (!Accept(TokenType.Operation, ";"))
            la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 3));
        GetNextToken();
        while (true)
        {
            try
            {
                Var();
            }
            catch
            {
                break;
            }
            if (!Accept(TokenType.Operation, ";"))
                la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 3));
            GetNextToken();
        }
    }
}

```

```

void Var() // <описание однотипных переменных>::=<имя>{,<имя>}<тип>
{
    if (curToken.tt != TokenType.Identifier)
        throw new Exception();

    /*имена переменных*/

    ctokenIdentifierNames.Add(((CTokenIdentifier)curToken).name);

    GetNextToken();

while (Accept(TokenType.Operation, ","))
{
    GetNextToken();
    if (curToken.tt != TokenType.Identifier)
        la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 2));
    /*имена переменных*/
    ctokenIdentifierNames.Add(((CTokenIdentifier)curToken).name);

    GetNextToken();
}

if (!Accept(TokenType.Operation, ":"))
    la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 3));

GetNextToken();

if (curToken.tt != TokenType.Operation)
    la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 3));

CType t;
/*добавление в таблицу имен*/
int code = ((CTokenOperation)curToken).code;
if (code == LexicalAnalyzer.keywords["integer"]) t = new CTypeInt();
else if (code == LexicalAnalyzer.keywords["real"]) t = new CTypeReal();
else if (code == LexicalAnalyzer.keywords["string"]) t = new CTypeString();
else if (code == LexicalAnalyzer.keywords["bool"]) t = new CTypeBoolean();
else throw new Exception();

foreach (string egge in ctokenIdentifierNames)
{
    tableOfIdentifiers.Add(egge, t);
}
ctokenIdentifierNames.Clear();
GetNextToken();
}

```

```

void Operator() //<оператор>::=<простой оператор>|<сложный оператор>
{

    if (curToken is CTokenOperation)
    {
        int code = ((CTokenOperation)curToken).code;
        if (code == LexicalAnalyzer.keyWords["begin"])
        {
            CompositeOperator();
            return;
        }
        else
        {
            if (code == LexicalAnalyzer.keyWords["while"])
            {
                CycleOpertor(); return;
            }
            else
            {
                if (code == LexicalAnalyzer.keyWords["if"])
                {
                    ChoiseOperator();
                    return;
                }
                else
                {
                    if (code == LexicalAnalyzer.keyWords["writeln"])
                    {
                        Write();
                        return;
                    }
                }
            }

            SimpleOperator();
        }
    }
}

```

```

void Write() // вывод ::= <writeln>'(<выражение>')'> ;
{
    GetNextToken();
    if (!Accept(TokenType.Operation, "("))
        la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 3));
    GetNextToken();

    var a = Expression();

    if (!Accept(TokenType.Operation, "="))
        la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 16));

    GetNextToken();
}

```

```

void SimpleOperator() // <простой оператор>::=<оператор присваивания>|<пустой оператор>
{
    try
    {
        AssignmentOperator();
    }
    catch
    {
    }
}

```

Ссылка: 1

```

void AssignmentOperator() // <оператор присваивания>::=<имя>:=<выражение>
{
    if (curToken.tt != TokenType.Identifier) {
        throw new Exception(); }

    CType left = tableOfIdentifiers[(curToken as CTokenIdentifier).name];

    GetNextToken();

    if (!Accept(TokenType.Operation, ":="))
        la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 3));

    GetNextToken();

    CType right = Expression();

    isDerivetToAssignemt(left, right);
}

```

```

CType Expression() // <выражение>::=<простое выражение>|<простое выражение><операция отношения><простое выражение>
{
    CType left = SimpleExpression();
    if (Realat())
    {
        GetNextToken();
        CType right = SimpleExpression();
        if (!left.isDerivedTo(right))
        {
            la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 14));
            GetNextToken();
        }
        return new CTypeBoolean();
    }
    return left;
}

```

```

CType SimpleExpression() // <простое выражение>::=<знак><слагаемое>{<аддитивная операция><слагаемое>}
{
    if (Accept(TokenType.Operation, "-"))
        GetNextToken();

    CType left = Term();

    while (true)
    {
        if (Add())
        {
            GetNextToken();
            CType right = Term();

            if (!left.isDerivedTo(right))
            {
                la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 14));
                GetNextToken();
            }
            left = DerivetTo(left, right);
        }
        else
        {
            break;
        }
    }
    return left;
}

```

```

CType Term() // <слагаемое>::=<множитель>{<мультипликативная операция><множитель>}
{
    CType left = Factor();
    while (true)
    {
        if (Mult())
        {
            GetNextToken();
            CType right = Factor();
            if (!left.isDerivedTo(right))
            {
                la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 7));
                GetNextToken();
            }
            left = DerivetTo(left, right);
        }
        else
        {
            break;
        }
    }
    return left;
}

```

```

bool Mult()
{
    if ((curToken is CTokenOperation) && ((curToken as CTokenOperation).code == LexicalAnalyzer.keyWords["*"] ||
        (curToken as CTokenOperation).code == LexicalAnalyzer.keyWords["/"] ||
        (curToken as CTokenOperation).code == LexicalAnalyzer.keyWords["div"] ||
        (curToken as CTokenOperation).code == LexicalAnalyzer.keyWords["mod"] ||
        (curToken as CTokenOperation).code == LexicalAnalyzer.keyWords["and"])))
    {
        return true;
    }
    return false;
}

```

Ссылка: 1

```

bool Add() // <аддитивная операция>::=+|-|or *
{
    if (curToken.tt == TokenType.Operation)
    {
        if ((curToken as CTokenOperation).code == LexicalAnalyzer.keyWords["+"] ||
            (curToken as CTokenOperation).code == LexicalAnalyzer.keyWords["-"] ||
            (curToken as CTokenOperation).code == LexicalAnalyzer.keyWords["or"])
        {
            return true;
        }
    }
    return false;
}

```



```

bool Realat() // <операция отношения>:==|<>|<|<=|>=|>|in      *
{
    if (curToken.tt == TokenType.Operation)
    {
        if ((curToken as CTokenOperation).code == LexicalAnalyzer.keyWords["="] ||
            (curToken as CTokenOperation).code == LexicalAnalyzer.keyWords["<>"] ||
            (curToken as CTokenOperation).code == LexicalAnalyzer.keyWords["<"] ||
            (curToken as CTokenOperation).code == LexicalAnalyzer.keyWords[">"] ||
            (curToken as CTokenOperation).code == LexicalAnalyzer.keyWords[">="] ||
            (curToken as CTokenOperation).code == LexicalAnalyzer.keyWords["<="] ||
            (curToken as CTokenOperation).code == LexicalAnalyzer.keyWords["in"])
        {
            return true;
        }
    }
    return false;
}

```

```

CType Factor() //<множитель>::=<имя>|<константа без знака>|(<выражение>)
{
    if (curToken.tt == TokenType.Identifier)
    {
        CType a = tableOfIdentifiers[(curToken as CTokenIdentifier).name];
        GetNextToken();
        if (a != null)
            return a;
        else
            la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 12));
    }

    if (curToken.tt == TokenType.Value)
    {
        if (((CTokenValue)curToken).value is CIntVariant)
        {
            GetNextToken();
            return new CTypeInt();
        }
        if (((CTokenValue)curToken).value is CRealVariant)
        {
            GetNextToken();
            return new CTypeReal();
        }
        if (((CTokenValue)curToken).value is CStringVariant)
        {
            GetNextToken();
            return new CTypeString();
        }
        if (((CTokenValue)curToken).value is CBooleanVariant)
        {
            GetNextToken();
            return new CTypeBoolean();
        }
    }

    return Expression();
    la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 9));
    GetNextToken();
    //return C_Expression();
}

```

```

void ChoiseOperator() // <выбирающий оператор>:= if <выражение> then <оператор>|if <выражение> then <оператор> else <оператор>
{
    GetNextToken();

    CType left = Expression();
    if (!(left is CTypeBoolean))
        la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 9));

    if (!Accept(TokenType.Operation, "then"))
        la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 3));

    GetNextToken();

    Operator();

    if (Accept(TokenType.Operation, "else"))
    {
        GetNextToken();
        Operator();
    }
}

Ссылка 1
void CycleOpertor() // <оператор цикла>:= while <выражение> do <оператор>
{
    GetNextToken();

    CType left = Expression();

    if (!(left is CTypeBoolean))
        la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 9));

    if (!Accept(TokenType.Operation, "do"))
        la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 3));
    GetNextToken();

    Operator();
}


```

Тестирование

Работа с простыми арифметическими операторами и сложными операторами

1) Входные данные:

Есть ошибка – присваиваем константе пустое значение

 file.dat – Блокнот


```

Файл  Правка  Формат  Вид  Справка
program hello;  {комментарий}
const A = ;
var A, B, C: integer;
begin
if A<>2 then A:=1;
A:=2+B;
end.

```

Ожидаемый результат: сообщение об ошибке

Выходной результат:

 output.txt – Блокнот

Файл Правка Формат Вид Справка


```
program hello; {комментарий}
const A = ;
Ошибка! Код: 9
Недопустимый тип

var A, B, C: integer;
begin
if A<>2 then A:=1;
A:=2+B;
end.
```

2) Входные данные:

Работа с простыми арифметическими операторами

Работа с оператором ветвления

 file.dat – Блокнот

Файл Правка Формат Вид Справка

```
program hello; {комментарий}
var A, B, C: integer;
begin
if A<>2 then A:=1;
A:=2+B;
end.
```

Ожидаемый результат: отсутствие ошибок и корректная работа программы

Выходной результат:

```
output.txt – Блокнот
Файл  Правка  Формат  Вид  Справка
program hello; {комментарий}
var A, B, C: integer;
begin
if A<>2 then A:=1;
A:=2+B;
end.
```

3) Входной результат:

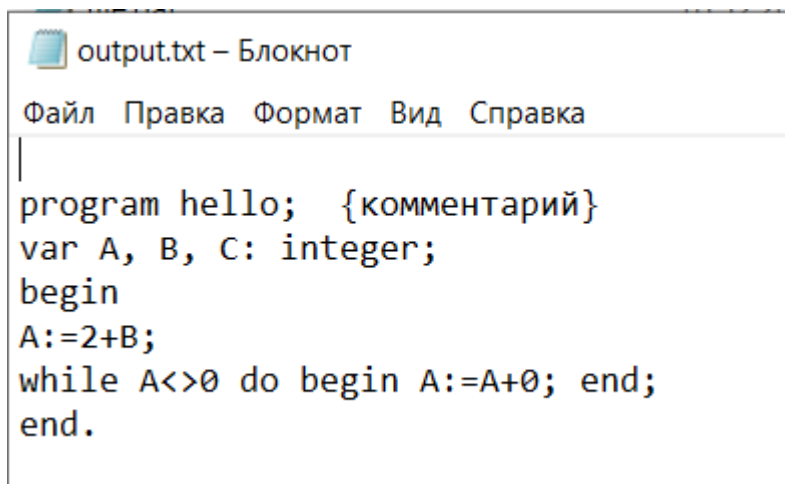
Работа с оператором цикла

Входной результат:

```
*file.dat – Блокнот
Файл  Правка  Формат  Вид  Справка
program hello; {комментарий}
var A, B, C: integer;
begin
A:=2+B;
while A<>0 do begin A:=A+0; end;
end.
```

Ожидаемый результат: отсутствие ошибок и корректная работа программы

Выходной результат:

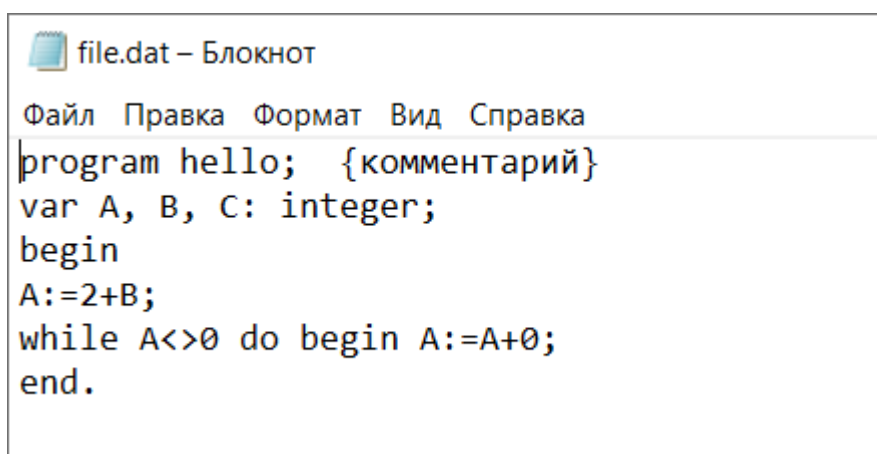


```
output.txt – Блокнот
Файл  Правка  Формат  Вид  Справка

program hello;  {комментарий}
var A, B, C: integer;
begin
A:=2+B;
while A<>0 do begin A:=A+0; end;
end.
```

4) Входной результат:

Есть синтаксическая ошибка – не хватает «end;»




```
file.dat – Блокнот
Файл  Правка  Формат  Вид  Справка

program hello;  {комментарий}
var A, B, C: integer;
begin
A:=2+B;
while A<>0 do begin A:=A+0;
end.
```

Ожидаемый результат: сообщение об ошибке

Выходной результат:

 output.txt – Блокнот

Файл Правка Формат Вид Справка

```
program hello; {комментарий}
var A, B, C: integer;
begin
A:=2+B;
Ошибка! Код: 2
Должно идти имя

while A<>0 do begin A:=A+0;
end.
Ошибка! Код: 2
Должно идти имя
```

Семантический анализатор

Описание

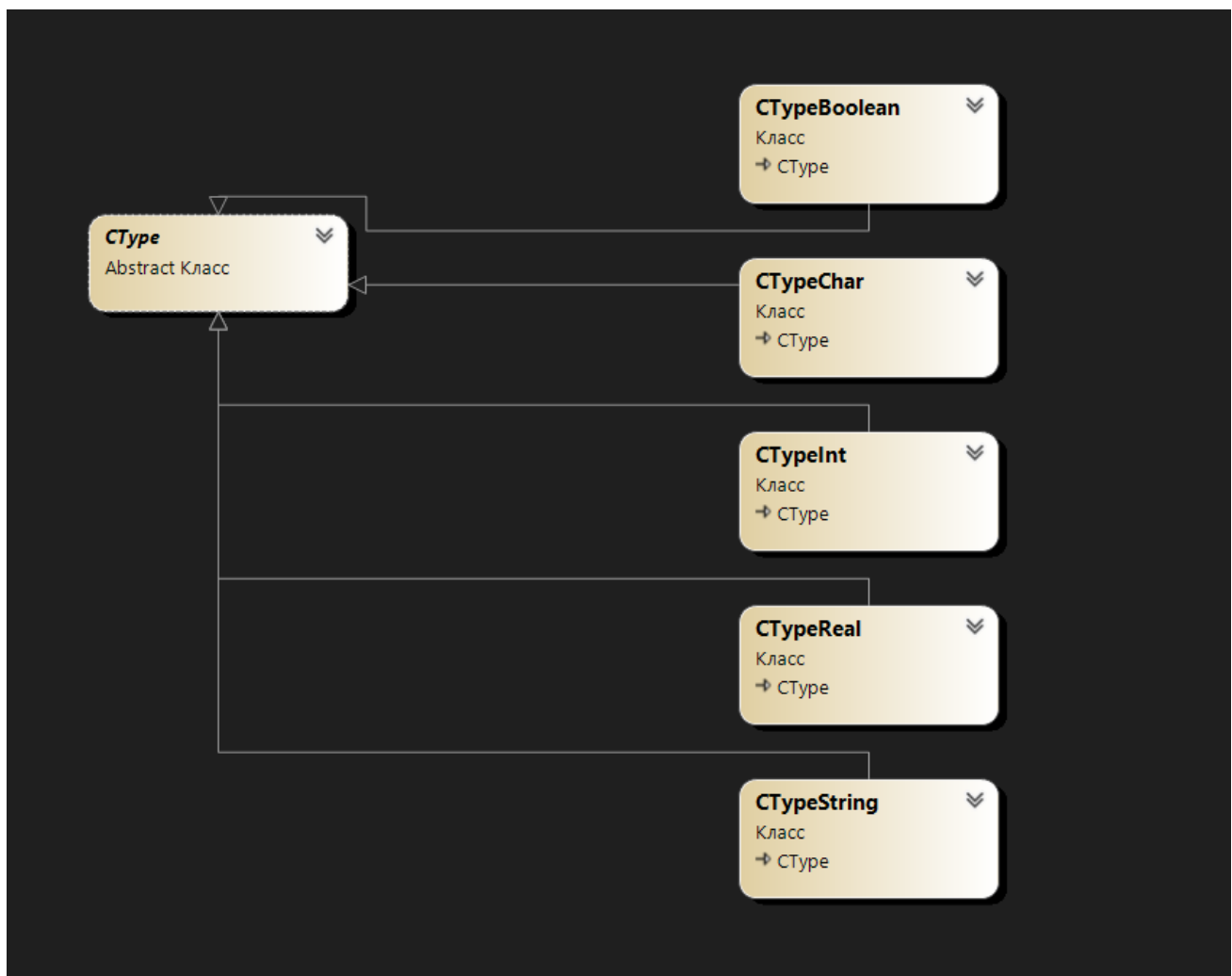
Семантический анализатор проверяет наличие соответствия типов, инициализацию всех переменных.

Проектирование

Нам необходимо проверять приводимость типов и инициализацию переменных.

Для этого нам нужно понимать какой тип возвращает оператор.

Были определены 5 основных типов:



Так же необходимо создать таблицу идентификаторов для хранения идентификатора и его типа.

Таблицы будет две, для хранения неописанных и описанных переменных.

Таблицу идентификаторов мы заполняем в блоке описания переменных.

Правила приводимости типов

1) Присваивание

- 1.1) Целому значению можно присвоить только целое значение
- 1.2) Вещественному значению можно присвоить целое или вещественное значение
- 1.3) Строковому типу можно присвоить только строковое значение
- 1.4) Логическому типу можно присвоить только логическое значение

2) Операции арифметические и логические

- 2.1) Проводить операции можно целыми и вещественным значениями
- 2.2) Остальные только с такими же типами

Реализация

Приводимость типов мы определяем по 2 действиям: присваивание и арифметические и логические операции.

Анализатор для арифметических операций хранится как метод для каждого класса отдельно

```
class CTypeInt : CType
{
    Ссылка: 3
    public CTypeInt() { }
    Ссылка: 4
    public override bool isDerivedTo(CType type)
    {
        return type is CTypeReal || type is CTypeInt;
    }
}
```

```

class CTypeReal : CType
{
    Ссылка: 4
    public CTypeReal() { }
    Ссылка: 4
    public override bool isDerivedTo(CType type)
    {
        return type is CTypeReal;
    }
}

```

```

class CTypeBoolean : CType
{
    Ссылка: 4
    public CTypeBoolean() { }
    Ссылка: 4
    public override bool isDerivedTo(CType type)
    {
        return type is CTypeBoolean;
    }
}

```

```

class CTypeString : CType
{
    Ссылка: 3
    public CTypeString() { }
    Ссылка: 4
    public override bool isDerivedTo(CType type)
    {
        return type is CTypeString;
    }
}

```

```

class CTypeChar : CType
{
    Ссылка: 1
    public CTypeChar() { }
    Ссылка: 4
    public override bool isDerivedTo(CType type)
    {
        return type is CTypeChar;
    }
}

```

А приводимость для присваивания работает как метод анализатора

```

public void isDerivetToAssignemt(CType left, CType right)
{
    if (left is CTypeInt && right is CTypeInt)
        return;

    if (left is CTypeBoolean && right is CTypeBoolean)
        return;

    if (left is CTypeString && right is CTypeString)
        return;
    if (left is CTypeReal && (right is CTypeInt || right is CTypeReal))
        return;

    la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 14));
    GetNextToken();
}

```

Так же есть метод для приведения типов

```
public CType DerivetTo(CType left, CType right)
{
    if (left is CTypeInt && right is CTypeInt)
        return left;

    if (left is CTypeBoolean && right is CTypeBoolean)
        return left;

    if (left is CTypeString && right is CTypeString)
        return left;

    return new CTypeReal();
}
```

Так как все операции, связанные с типами, происходят в БНФ, связанных с операциями, они должны возвращать тип значения, с которым работают (<слагаемое>, <простое выражение>, <выражение>, <множитель>).

```

CType Factor() //<множитель>::=<имя>|<константа без знака>|(<выражение>)
{
    if (curToken.tt == TokenType.Identifier)
    {
        CType a = tableOfIdentifiers[(curToken as CTokenIdentifier).name];
        GetNextToken();
        if (a != null)
            return a;
        else
            la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 12));
    }

    if (curToken.tt == TokenType.Value)
    {
        if (((CTokenValue)curToken).value is CIntVariant)
        {
            GetNextToken();
            return new CTypeInt();
        }
        if (((CTokenValue)curToken).value is CRealVariant)
        {
            GetNextToken();
            return new CTypeReal();
        }
        if (((CTokenValue)curToken).value is CStringVariant)
        {
            GetNextToken();
            return new CTypeString();
        }
        if (((CTokenValue)curToken).value is CBooleanVariant)
        {
            GetNextToken();
            return new CTypeBoolean();
        }
    }

    return Expression();
    la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 9));
    GetNextToken();
    //return Expression();
}

```

```

CType Term() // <слагаемое>::=<множитель>{<мультипликативная операция><множитель>}
{
    CType left = Factor();
    while (true)
    {
        if (Mult())
        {
            GetNextToken();
            CType right = Factor();
            if (!left.isDerivedTo(right))
            {
                la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 7));
                GetNextToken();
            }
            left = DerivetTo(left, right);
        }
        else
        {
            break;
        }
    }
    return left;
}

```

```

CType SimpleExpression() // <простое выражение>::=<знак><слагаемое>{<аддитивная операция><слагаемое>}
{
    if (Accept(TokenType.Operation, "-"))
        GetNextToken();

    CType left = Term();

    while (true)
    {
        if (Add())
        {
            GetNextToken();
            CType right = Term();

            if (!left.isDerivedTo(right))
            {
                la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 14));
                GetNextToken();
            }
            left = DerivetTo(left, right);
        }
        else
        {
            break;
        }
    }
    return left;
}

```

```

CType Expression() // <выражение>::=<простое выражение>|<простое выражение><операция отношения><простое выражение>
{
    CType left = SimpleExpression();
    if (Realat())
    {
        GetNextToken();
        CType right = SimpleExpression();
        if (!left.isDerivedTo(right))
        {
            la.IO.errorsInLine.Add(new Error(la.IO.curPosition, 14));
            GetNextToken();
        }
        return new CTypeBoolean();
    }
    return left;
}

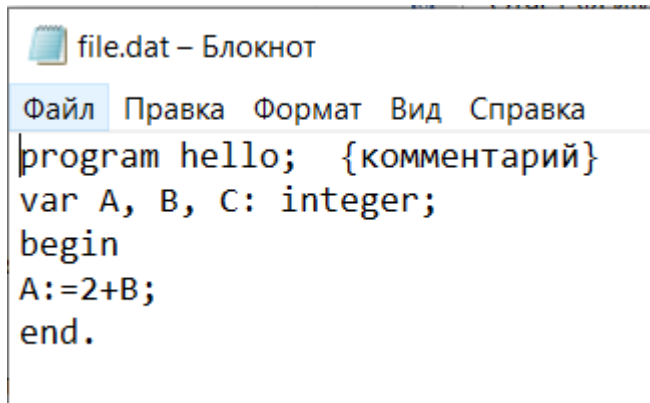
```

Проверка типа возвращенного нижней БНФ происходит в вызвавшей БНФ.

Тестирование

Проверка приводимости типов при арифметической операции и присваивания

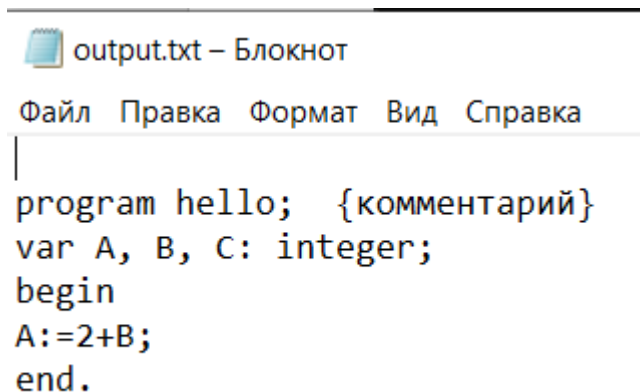
1) Входные данные:



```
file.dat – Блокнот
Файл  Правка  Формат  Вид  Справка
program hello; {комментарий}
var A, B, C: integer;
begin
A:=2+B;
end.
```

Ожидаемый результат: Отсутствие ошибок, нормальная работа программы без прерываний


Выходной результат:



```
output.txt – Блокнот
Файл  Правка  Формат  Вид  Справка
|
program hello; {комментарий}
var A, B, C: integer;
begin
A:=2+B;
end.
```

Проверка приводимости типов при арифметической операции и присваивания, и логической операции

2) Входные данные:


 file.dat – Блокнот

Файл Правка Формат Вид Справка

```
program hello; {комментарий}
var A, B, C: integer;
begin
A:=2+B;
if A<>2 then A:=2;
end.
```

Ожидаемый результат: Отсутствие ошибок, нормальная работа программы без прерываний

Выходные данные:

 output.txt – Блокнот


Файл Правка Формат Вид Справка

```

program hello; {комментарий}
var A, B, C: integer;
begin
A:=2+B;
if A<>2 then A:=2;
end.
```

Ошибка в приведении типа (проверка нейтрализации ошибок)


3) Входные данные:

 file.dat – Блокнот

Файл Правка Формат Вид Справка
program hello; {комментарий}
var A, B, C: integer;
D: string;
begin
A:=2+B;
if A<>2 then A:=2+D;
end.

Ожидаемый результат: ошибка не приводимости типов

Выходной результат:

 output.txt – Блокнот

Файл Правка Формат Вид Справка
program hello; {комментарий}
var A, B, C: integer;
D: string;
begin
A:=2+B;
if A<>2 then A:=2+D;
Ошибка! Код: 14
Несоответствие типов

end.
Ошибка! Код: 14
Несоответствие типов