I took a programming language using ANTLR 4 as a project for this semester. Here is a documentation on how to use it, how to check for bugs and explanation of the code.

1. **A plan for testing the code.**

In the file language.ls there is a code, written using rules that are defined for my programming language:

- a = 5 – assignment
- While ___ do {} – while loop (could be While ___ do {} Else do {})
- If ___ do {} – if condition (could be If ___ do {} Else do {}
- Some arithmetical, logical and comparison operators:

```
mulOp: '*' | '/' | '%' ;
addOp: '+' | '-' ;
compOp: '==' | '!=' | '>' | '>=' | '<' | '<=' ;
boolOp: 'and' | 'or' | 'xor' ;
```

My language is some mix of Python and C#. From Python, you do not need to put any semi-columns at the end of expression, you do not need to say which type of variable you assign. From C#, you have {block} after While ___ do, If __ do and Else, you have two functions that input something to the console:

- Write(___) or Write(___ , ___ , …) in case you need to output some data in one line
- WriteLn(___) or WriteLn(___ , ___ , …) in case you need to output each piece of data on the new line

All While, If and Else statements must start with a capital letter.

If you want to try to write something yourself, be careful, as language is very case-sensitive:

- You cannot write (4.3 == 4.3) or (4.3 != 3.3) as machine rounding of floats will be less accurate
- Strings accept only “+” operator (regular concatenation) and do not support any other operations (“abc” < “bbc”)

2. **Proper documentation of the code, including clear and descriptive comments.**

ANTLR 4 is a package that is being used to define a programming language: the grammar and the structure of it. I called my programming language “language”. In my project I have 4 files, namely:

- **Class.cs**
- **LanguageVisitor.cs**
- **language.g4 (g4 is an extension used for file that is using ANTLR 4 to define a language)**
- **test.ls (ls states for language script)**

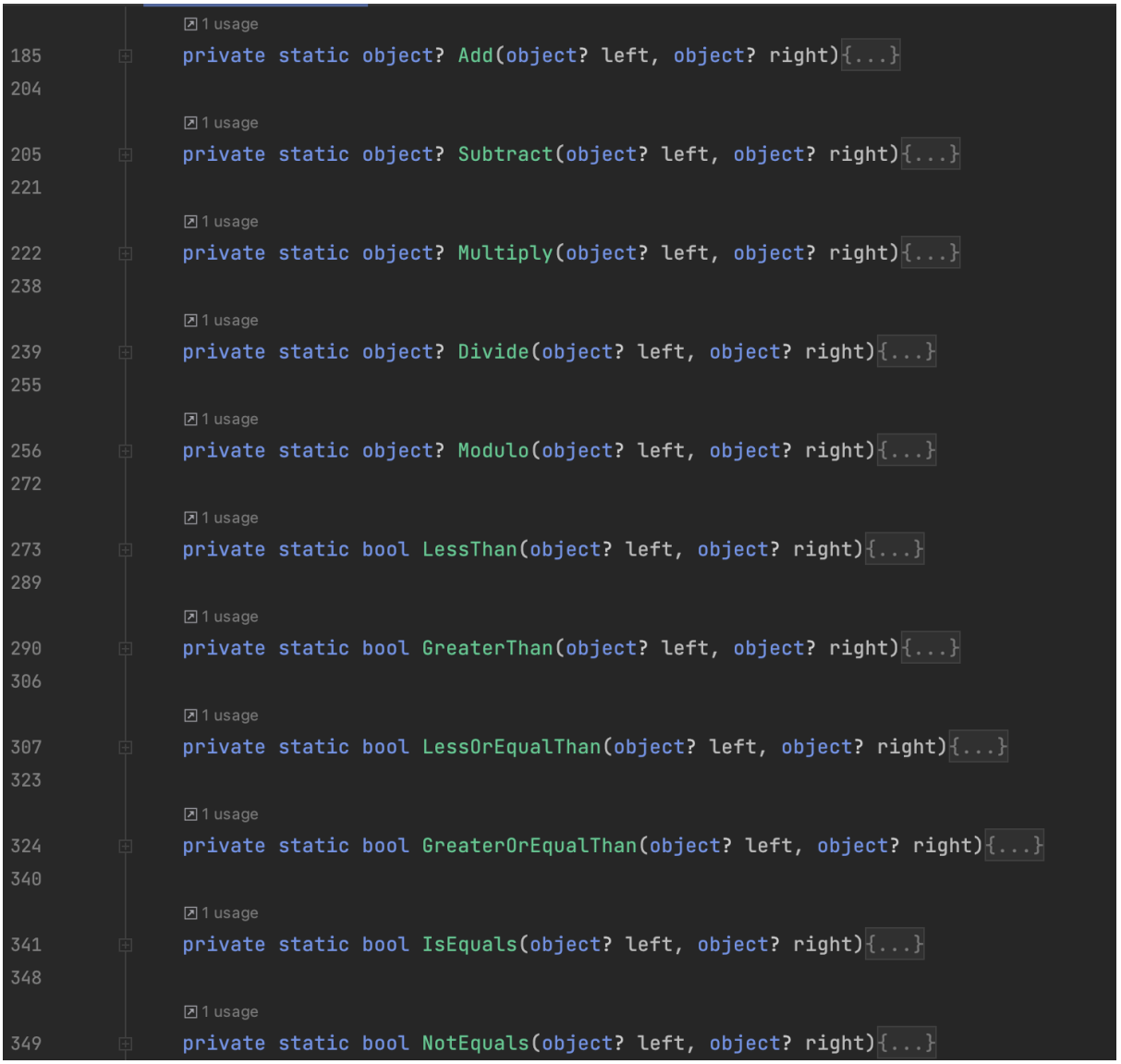
Two last files are in the folder named “Content”.

In **Class.cs** there are a bunch of lines that show how to create the lexer and then create the tree (lines 8-11). The subsequent lines show how to launch the visitor that you have created: you have to get the context for whichever starting rule you use, in our case program, and the order to visit the tree from that node.

In **LanguageVisitor.cs** I override methods from original languageBaseVisitor class, so they work as I want:

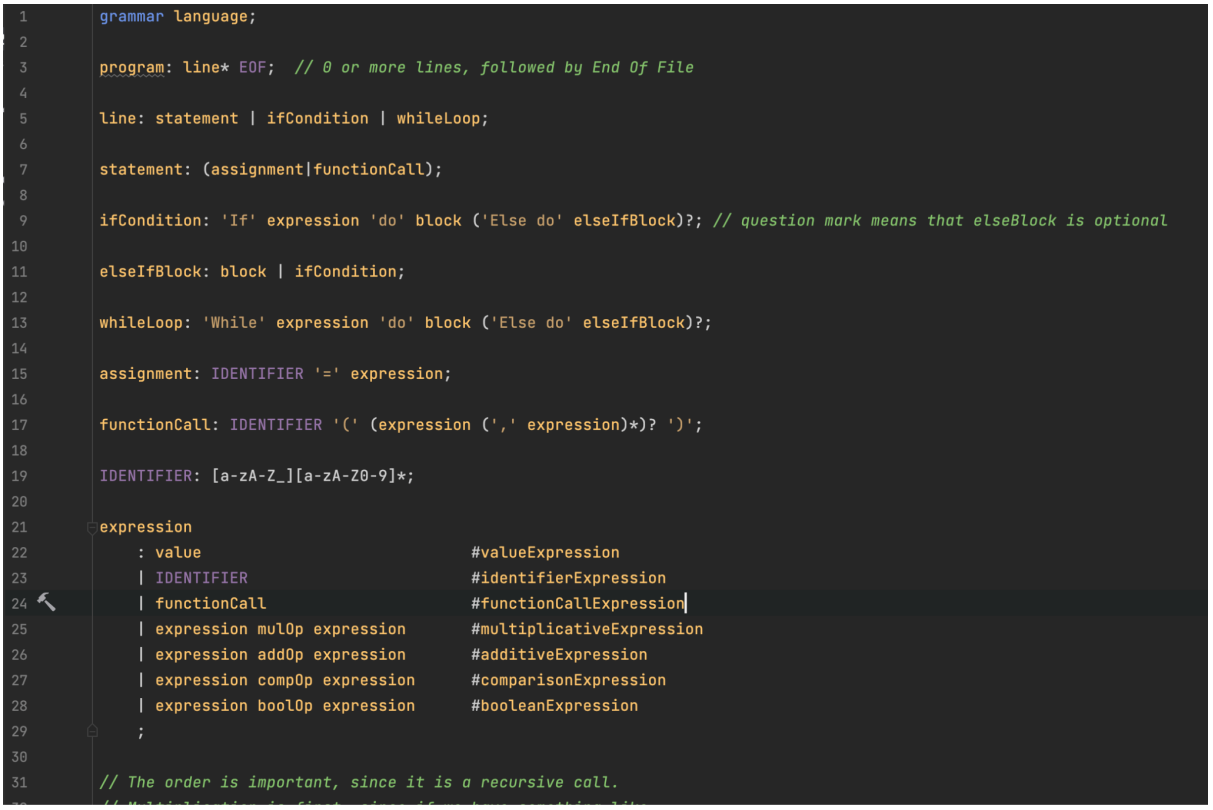
```
public class LanguageVisitor : languageBaseVisitor<object?>
{
    7 usages
    private Dictionary<string, object?> Variables { get; } = new();

    1 usage
    public LanguageVisitor(){...}

    0+1 usages
    public override object? VisitFunctionCall(languageParser.FunctionCallContext context){...}

    1 usage
    private static object? Write(object?[] args){...}

    1 usage
    private static object? WriteLn(object?[] args){...}

    0+1 usages
    public override object? VisitAssignment(languageParser.AssignmentContext context){...}

    0+1 usages
    public override object? VisitIdentifierExpression(languageParser.IdentifierExpressionContext context){...}

    0+1 usages
    public override object? VisitValue(languageParser.ValueContext context){...}

    0+1 usages
    public override object? VisitMultiplicativeExpression(languageParser.MultiplicativeExpressionContext context){...}
```

Lines from 185 to 387 are some helper methods as:



Where I consider cases of using arithmetical, logical and comparison operators on different types of variables

In **language.g4** I define my language:



```
grammar language;

program: line* EOF;  // 0 or more lines, followed by End Of File

line: statement | ifCondition | whileLoop;

statement: (assignment|functionCall);

ifCondition: 'If' expression 'do' block ('Else do' elseIfBlock)?; // question mark means that elseBlock is optional

elseIfBlock: block | ifCondition;

whileLoop: 'While' expression 'do' block ('Else do' elseIfBlock)?;

assignment: IDENTIFIER '=' expression;

functionCall: IDENTIFIER '(' (expression (',' expression)*)? ')';

IDENTIFIER: [a-zA-Z_][a-zA-Z0-9]*;

expression
    : value                          #valueExpression
    | IDENTIFIER                     #identifierExpression
    | functionCall                   #functionCallExpression
    | expression mulOp expression    #multiplicativeExpression
    | expression addOp expression    #additiveExpression
    | expression compOp expression   #comparisonExpression
    | expression boolOp expression   #booleanExpression
    ;

// The order is important, since it is a recursive call.
```

There are a bunch of comments in the code that I used to clarify what or why I wrote exactly like that.

3. **A user manual for any user-facing aspects of the project, such as a GUI or command-line interface.**

Since it is a programming language, you can use it by simply writing a test program using rules and limitations, described above. Here is a sample program and output to it:

Sample program:

```
a = 9

Write("a is ", a)



Write("a + 5 = ", a + 5)

Write("a + 5.3 = ", a + 5.3)

Write("a - 5 = ", a - 5)

Write("a * 5 = ", a * 5)

Write("a / 3 = ", a / 3)

Write("a % 5 = ", a % 5)



While a > 6 do

{

   a = a - 2

   Write("a is ", a)

}



If a == 5 do

{

   WriteLn("a is equals 5")

}



If a == 5 and 7 + 1 == 8 do

{
```

```
   WriteLn("Both conditions are true")

}



If a == 4 or 7 + 1 == 9 do

{

   WriteLn("Only one condition is true")

}

Else do

{

   WriteLn("Either both or none of conditions are true")

}



Write("a > 4 is", a > 4)

Write("a < 4 is", a < 4)

Write("a != 4 is", a != 4)



name = 'Alex'

fullname = 'Oleksandr'

Write("My name is ", name)

surname = 'Multykh'

Write("My surname is ", surname)

Write("My full name is ", fullname + " " + surname)

WriteLn(name > surname)                                    // error
```

Sample output:

```
a is 9
a + 5 = 14
a + 5.3 = 14,3
a - 5 = 4
a * 5 = 45
a / 3 = 3
a % 5 = 4
a is 7
a is 5
a is equals 5
Both conditions are true
Eiter both or none of conditions are true
a > 4 isTrue
a < 4 isFalse
a != 4 isTrue
My name is Alex
My surname is Multykh
My full name is Oleksandr Multykh
Unhandled exception. System.Exception: Cannot compare values of types System.String and System.String.
```

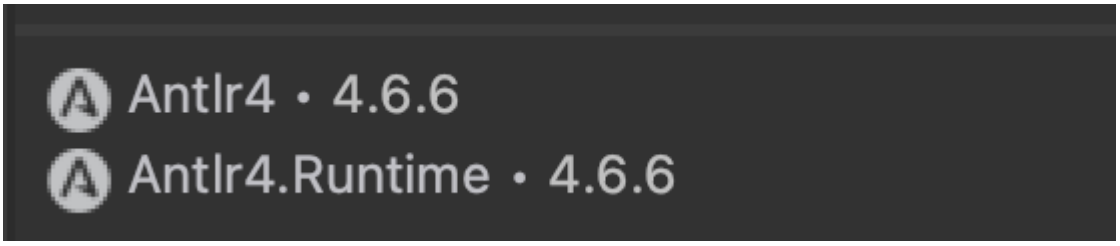
4. **A final report summarizing the project and its results.**

1. Introduction

My programming language was chosen as a project to practice this kind of using C# and practice using external packages for it. I thought it would be interesting and knowledgeable to write such code and it really was.

2. Project Requirements

My programming language project is quite easy to go through. The only things needed for running it are following two external packages:



Then, when you have installed them, you simply build a solution if my project and run it

3. Methodology

I have used several methods to write the code:

- Firstly, I have looked through the documentation of ANTLR 4 to understand how to use it.
- Secondly, I have created all needed files and defined my language in **language.g4**..

- Thirdly, I have added code to the **Class.cs** that was needed to create the lexer and then create the tree.
- Fourthly, I have overridden the class languageBaseVisitor in **LanguageVisitor.cs** and added all helper methods that were mentioned above.
- The most challenging part for me was to understand the nuances and details of ANTLR 4 and to apply that knowledge to override the class languageBaseVisitor.

4. Results and Outcomes

The result of my programming language project is surprisingly a working programming language that accepts assignment of variables of types int, float, string and bool, While Else loops, If Else statements and basic arithmetical, logical and comparison operators.

While writing my code, I have been using only these things:

- Adam Dingle's lecture notes
- C# official documentation provided by Microsoft
- Official ANTLR 4 documentation and a small article on how to start working with ANTLR 4, I found on the internet

While making this programming language project, I learned to interact with C# and ANTLR 4, to overcome difficulties and bugs when coding, to make a working programming language using ANTLR 4, that can be extended in the future to support more complicated operations and to learn how to apply external packages I have never heard of before to my project using their documentation.