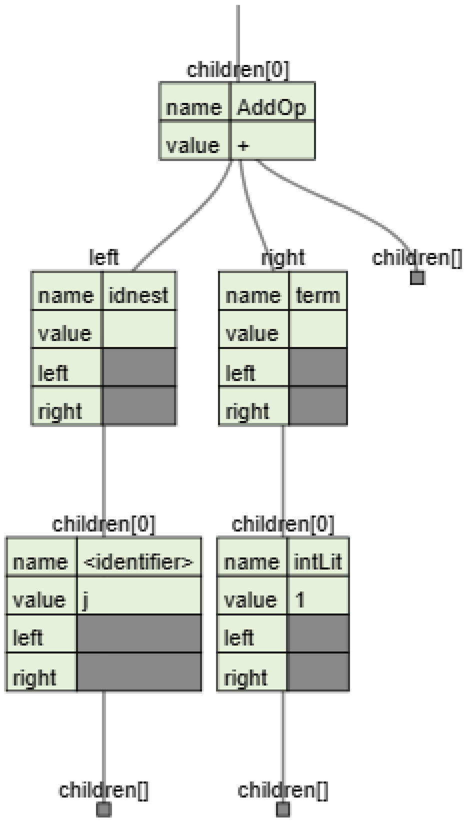
# Section 1. Attribute grammar :

Qiantongzhou40081938

**semantic actions:**

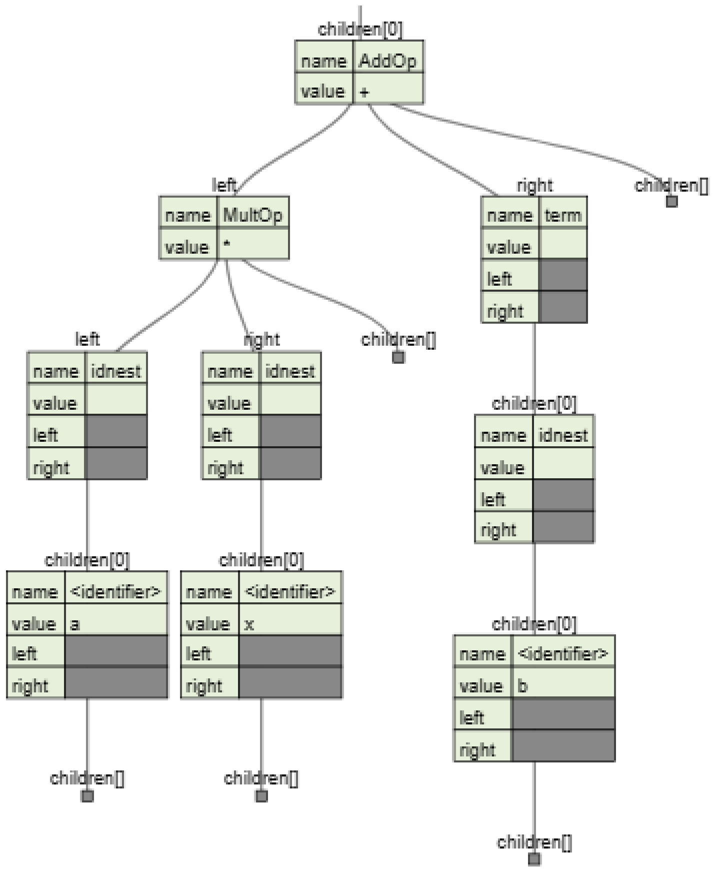
**AddOp:**

```
Expr -> ArithExpr { $$.value = $1.value; }

ArithExpr -> Term AddOp ArithExpr {

        if ($2.value == "+") {

                $$.value = $1.value + $3.value;

        } else if ($2.value == "-") {

                $$.value = $1.value - $3.value;

         }

}
```



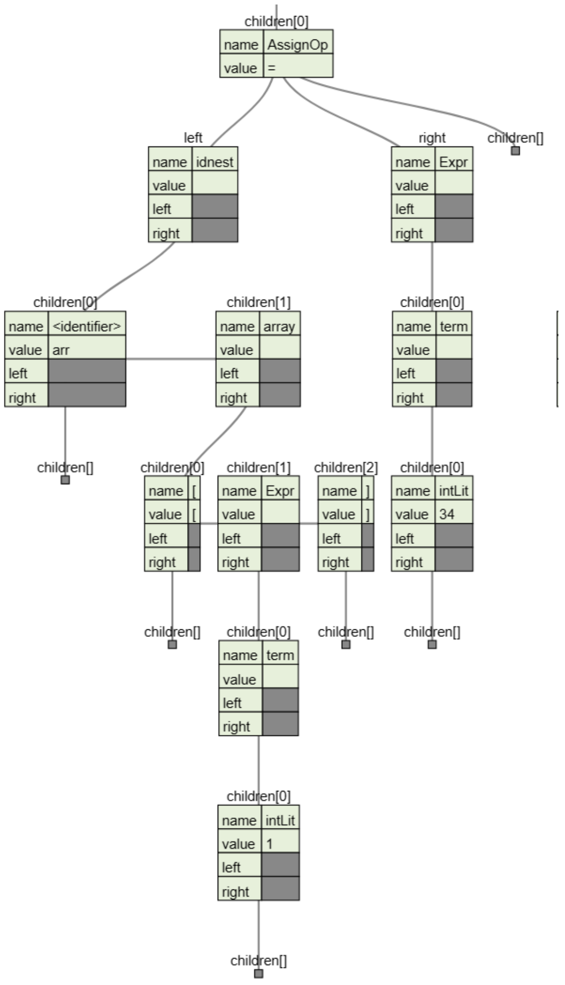
**MultiOp**:

```
ArithExpr -> Term { $$.value = $1.value; }
Term -> Factor MultOp Term {
        if ($2.value == "*") {
                $$.value = $1.value * $3.value;
        } else if ($2.value == "/") {
                $$.value = $1.value / $3.value;
        }
}
```



**AssignOp**:

```
        <AssignOp> ::= '='   { $$.value = $1.value; $$.type = $3.type; }

        <Expression> ::= <Variable> <AssignOp> <Expression>

        {

                if ($1.type != $3.type) {

                        // report type error

                        printf("Type error: cannot assign %s to %s\n", $3.type, $1.type);

                 } else {

                 $$.type = $1.type;

        }

}
```



**RelOp**:

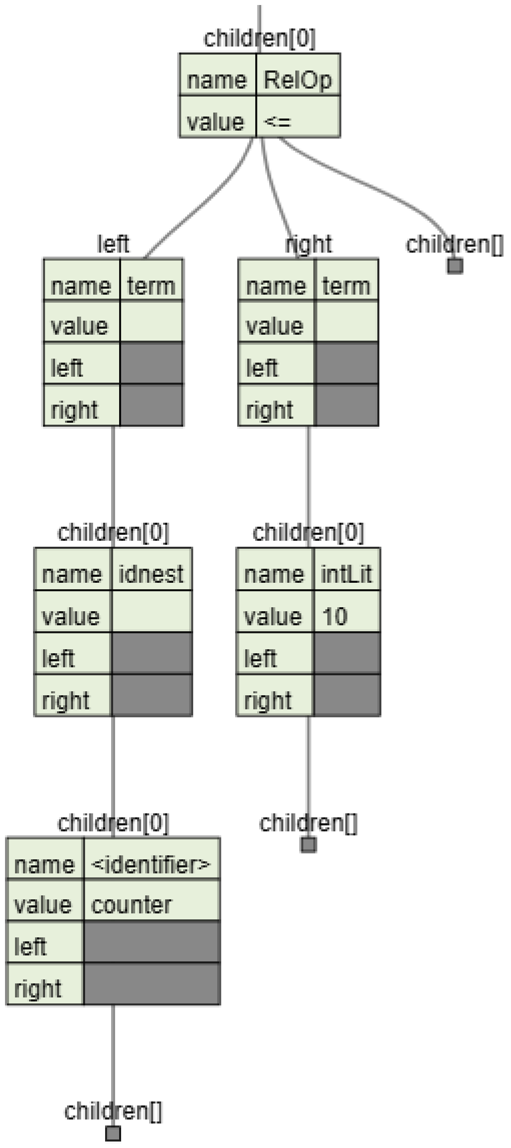
E -> E1 > E2 {E.value = (E1.value > E2.value) ? 1 : 0;}

E -> E1 < E2 {E.value = (E1.value < E2.value) ? 1 : 0;}

E -> E1 >= E2 {E.value = (E1.value >= E2.value) ? 1 : 0;}

E -> E1 <= E2 {E.value = (E1.value <= E2.value) ? 1 : 0;}

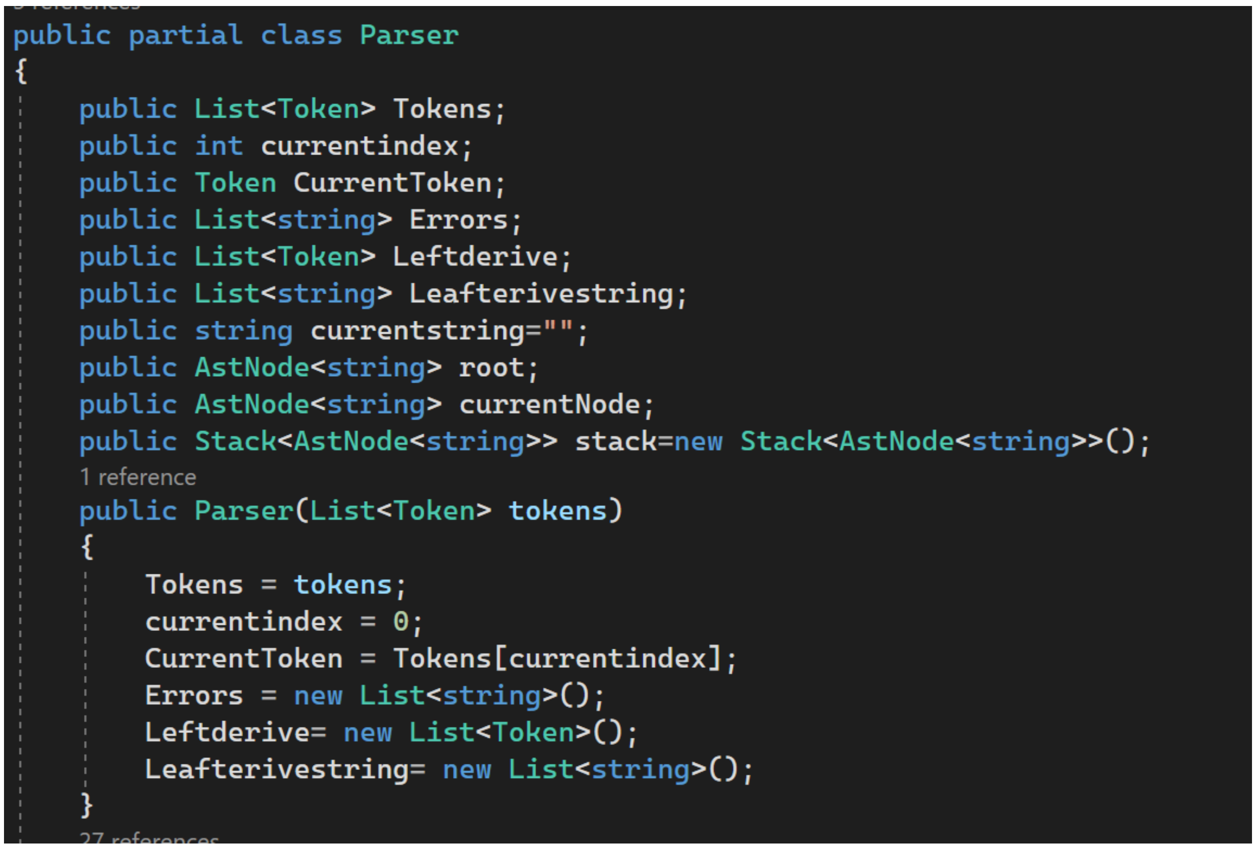
E -> E1 == E2 {E.value = (E1.value == E2.value) ? 1 : 0;}



## Section 2. Design :

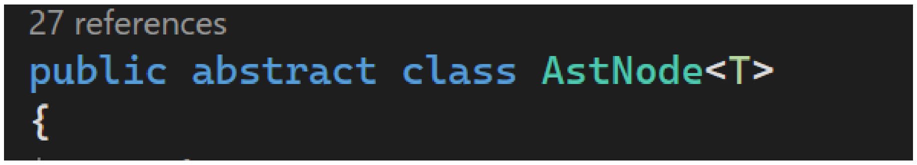
**This project implements a parser for a custom programming language and augments it with syntax-directed translation to generate an abstract syntax tree (AST) data structure. The project is divided into three main components:**

**Parser: The parser takes input source files written in the custom language and generates a parse tree. The parse tree is then transformed into an AST using syntax-directed translation. The AST represents the structure of the program in a way that is easy to manipulate and analyze.**



```
public partial class Parser
{
    public List<Token> Tokens;
    public int currentindex;
    public Token CurrentToken;
    public List<string> Errors;
    public List<Token> Leftderive;
    public List<string> Leafterivestring;
    public string currentstring="";
    public AstNode<string> root;
    public AstNode<string> currentNode;
    public Stack<AstNode<string>> stack=new Stack<AstNode<string>>();
    1 reference
    public Parser(List<Token> tokens)
    {
        Tokens = tokens;
        currentindex = 0;
        CurrentToken = Tokens[currentindex];
        Errors = new List<string>();
        Leftderive= new List<Token>();
        Leafterivestring= new List<string>();
    }
```

**ASTNode is a class that represents a node in an Abstract Syntax Tree (AST). An AST is a tree data structure that represents the abstract syntactic structure of a program. It is generated by parsing the source code of a program and represents the program in a way that is easier to process than the original source code.**



```
27 references
public abstract class AstNode<T>
{
```

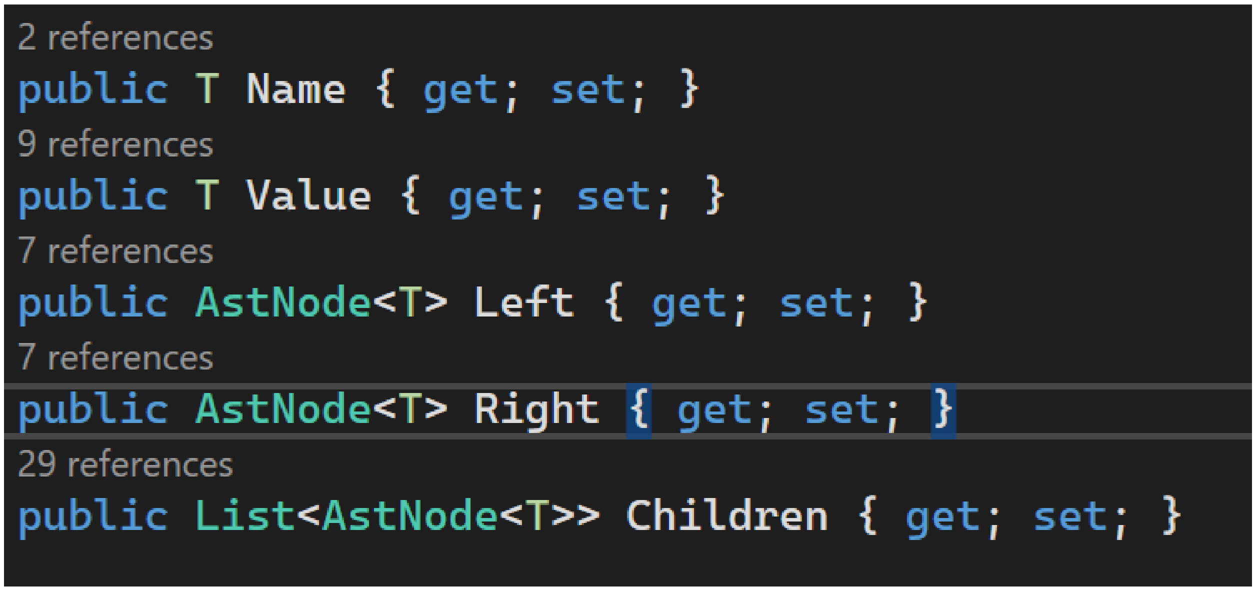
**The ASTNode class has four fields:**

**value:** represents the value of the node, which could be an operator, an identifier, a constant, or any other token in the program.

**children:** a list of child nodes of the current node. Each child node is also an instance of the ASTNode class.

**leftNode:** represents the left child node of the current node.

**rightNode**: represents the right child node of the current node.

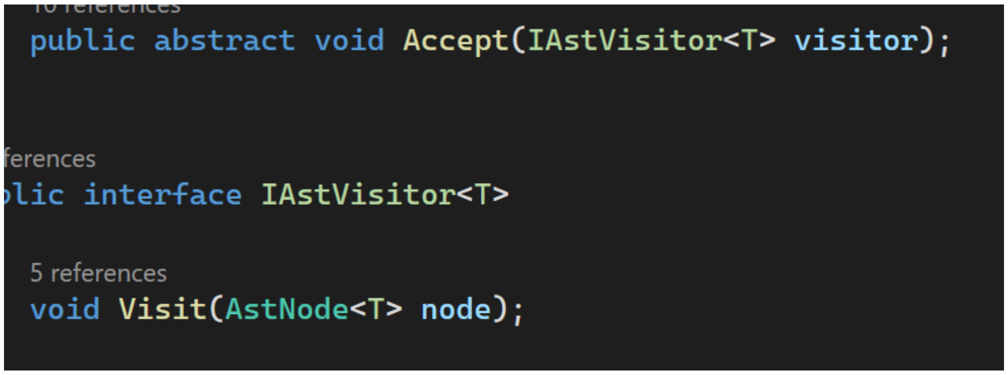


```
2 references
public T Name { get; set; }
9 references
public T Value { get; set; }
7 references
public AstNode<T> Left { get; set; }
7 references
public AstNode<T> Right { get; set; }
29 references
public List<AstNode<T>> Children { get; set; }
```

**The ASTNode class has several methods:**

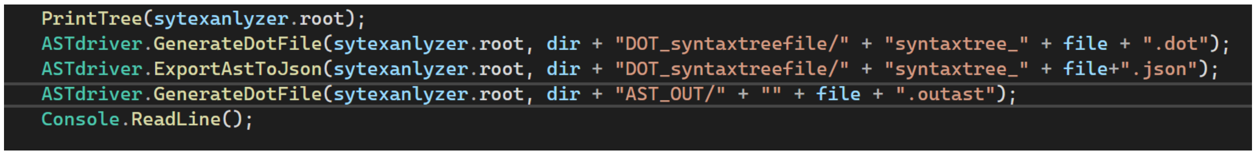
**addChild():** adds a child node to the current node.

**accept():** accepts a visitor object that can perform operations on the AST nodes. This is used to perform semantic analysis, code generation, and other tasks on the AST, interface for future extension.



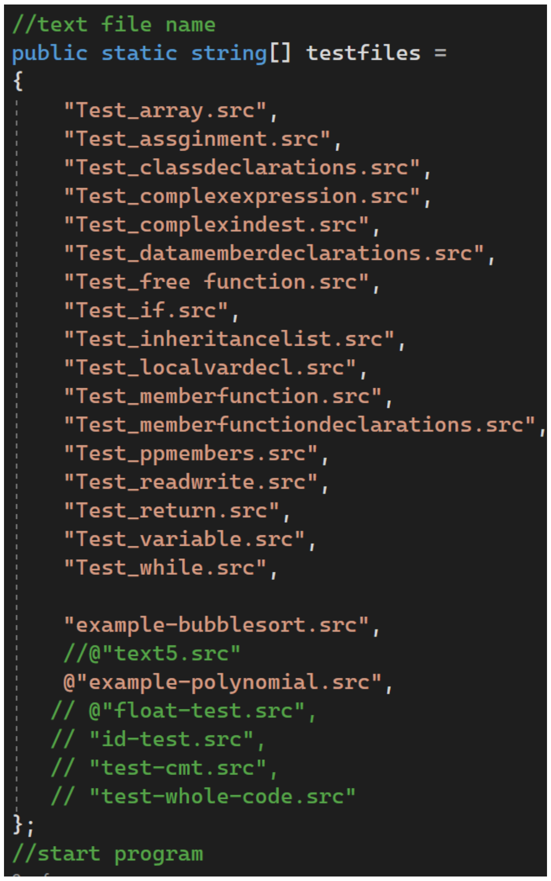
```
 10 references
 public abstract void Accept(IAstVisitor<T> visitor);


ferences
lic interface IAstVisitor<T>

 5 references
 void Visit(AstNode<T> node);
```

**AST Output: The generated AST is outputted to a file in a format that allows easy visualization of the structure of the tree. The output file is named after the original source file with an added ".outast" extension. The output file contains a text representation of the abstract syntax tree representing the original program.**



```
    PrintTree(sytexanlyzer.root);
    ASTdriver.GenerateDotFile(sytexanlyzer.root, dir + "DOT_syntaxtreefile/" + "syntaxtree_" + file + ".dot");
    ASTdriver.ExportAstToJson(sytexanlyzer.root, dir + "DOT_syntaxtreefile/" + "syntaxtree_" + file+".json");
    ASTdriver.GenerateDotFile(sytexanlyzer.root, dir + "AST_OUT/" + "" + file + ".outast");
    Console.ReadLine();
```

**Test Cases: A set of source files is provided that enables testing of the AST generation for all syntactical structures and sub-structures involved in the language. A driver program is included that parses all the test files and generates corresponding output files.**



```
//text file name
public static string[] testfiles =
{
    "Test_array.src",
    "Test_assginment.src",
    "Test_classdeclarations.src",
    "Test_complexexpression.src",
    "Test_complexindest.src",
    "Test_datamemberdeclarations.src",
    "Test_free function.src",
    "Test_if.src",
    "Test_inheritancelist.src",
    "Test_localvardecl.src",
    "Test_memberfunction.src",
    "Test_memberfunctiondeclarations.src",
    "Test_ppmembers.src",
    "Test_readwrite.src",
    "Test_return.src",
    "Test_variable.src",
    "Test_while.src",

    "example-bubblesort.src",
    //@"text5.src"
    @"example-polynomial.src",
   // @"float-test.src",
   // "id-test.src",
   // "test-cmt.src",
   // "test-whole-code.src"
};
//start program
```

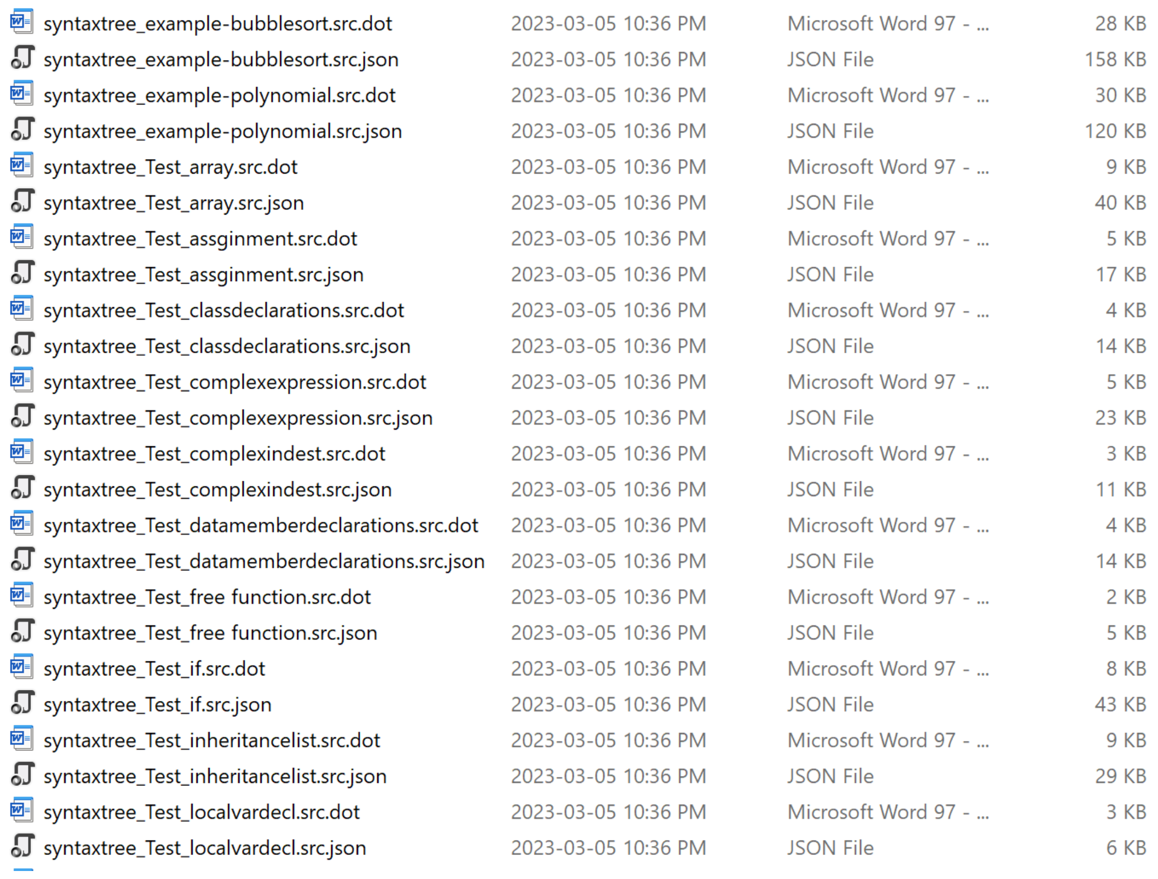
**The project is implemented in C#  and requires the following files to be submitted:**

**Source code files: The source code should be separated into modules using a comprehensible coding style.**

**Test files: A group of test files is included that enable testing of the AST generation for all syntactical structures and sub-structures involved in the language.**

**Report: A brief report in PDF format should be included.**

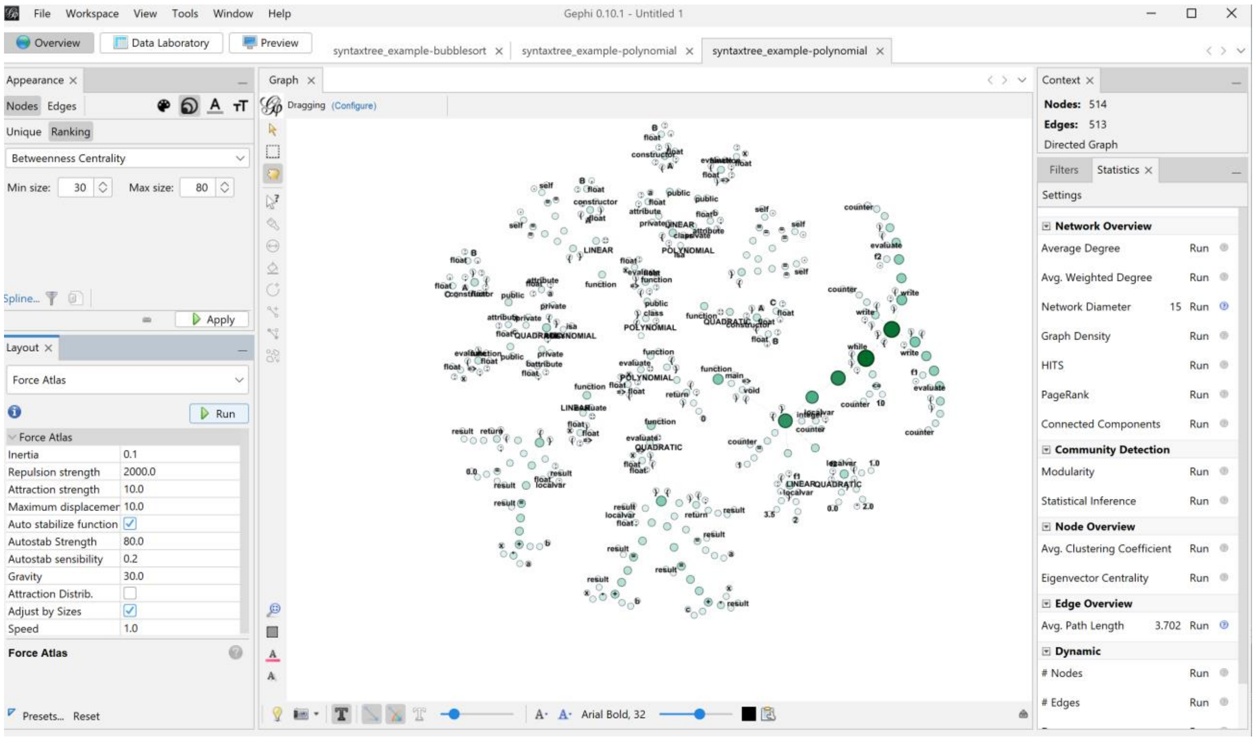
**OUTPUT: file for output: DOT, JSON, ASTOUT.**



| | | | |
|---|---|---|---|
| syntaxtree_example-bubblesort.src.dot | 2023-03-05 10:36 PM | Microsoft Word 97 - ... | 28 KB |
| syntaxtree_example-bubblesort.src.json | 2023-03-05 10:36 PM | JSON File | 158 KB |
| syntaxtree_example-polynomial.src.dot | 2023-03-05 10:36 PM | Microsoft Word 97 - ... | 30 KB |
| syntaxtree_example-polynomial.src.json | 2023-03-05 10:36 PM | JSON File | 120 KB |
| syntaxtree_Test_array.src.dot | 2023-03-05 10:36 PM | Microsoft Word 97 - ... | 9 KB |
| syntaxtree_Test_array.src.json | 2023-03-05 10:36 PM | JSON File | 40 KB |
| syntaxtree_Test_assginment.src.dot | 2023-03-05 10:36 PM | Microsoft Word 97 - ... | 5 KB |
| syntaxtree_Test_assginment.src.json | 2023-03-05 10:36 PM | JSON File | 17 KB |
| syntaxtree_Test_classdeclarations.src.dot | 2023-03-05 10:36 PM | Microsoft Word 97 - ... | 4 KB |
| syntaxtree_Test_classdeclarations.src.json | 2023-03-05 10:36 PM | JSON File | 14 KB |
| syntaxtree_Test_complexexpression.src.dot | 2023-03-05 10:36 PM | Microsoft Word 97 - ... | 5 KB |
| syntaxtree_Test_complexexpression.src.json | 2023-03-05 10:36 PM | JSON File | 23 KB |
| syntaxtree_Test_complexindest.src.dot | 2023-03-05 10:36 PM | Microsoft Word 97 - ... | 3 KB |
| syntaxtree_Test_complexindest.src.json | 2023-03-05 10:36 PM | JSON File | 11 KB |
| syntaxtree_Test_datamemberdeclarations.src.dot | 2023-03-05 10:36 PM | Microsoft Word 97 - ... | 4 KB |
| syntaxtree_Test_datamemberdeclarations.src.json | 2023-03-05 10:36 PM | JSON File | 14 KB |
| syntaxtree_Test_free function.src.dot | 2023-03-05 10:36 PM | Microsoft Word 97 - ... | 2 KB |
| syntaxtree_Test_free function.src.json | 2023-03-05 10:36 PM | JSON File | 5 KB |
| syntaxtree_Test_if.src.dot | 2023-03-05 10:36 PM | Microsoft Word 97 - ... | 8 KB |
| syntaxtree_Test_if.src.json | 2023-03-05 10:36 PM | JSON File | 43 KB |
| syntaxtree_Test_inheritancelist.src.dot | 2023-03-05 10:36 PM | Microsoft Word 97 - ... | 9 KB |
| syntaxtree_Test_inheritancelist.src.json | 2023-03-05 10:36 PM | JSON File | 29 KB |
| syntaxtree_Test_localvardecl.src.dot | 2023-03-05 10:36 PM | Microsoft Word 97 - ... | 3 KB |
| syntaxtree_Test_localvardecl.src.json | 2023-03-05 10:36 PM | JSON File | 6 KB |

**ASTdriver executable: An executable named "ASTdriver" should be included that extracts the tokens from all test files and generates corresponding outast files.**

## Section 3. Use of tools :



Betweenness Centrality
Min size: 30 Max size: 80

Force Atlas

| | |
|---|---|
| Inertia | 0.1 |
| Repulsion strength | 2000.0 |
| Attraction strength | 10.0 |
| Maximum displacement | 10.0 |
| Auto stabilize function | ✓ |
| Autostab Strength | 80.0 |
| Autostab sensibility | 0.2 |
| Gravity | 30.0 |
| Attraction Distrib. | |
| Adjust by Sizes | ✓ |
| Speed | 1.0 |

Nodes: 514
Edges: 513
Directed Graph

Network Overview
Average Degree
Avg. Weighted Degree
Network Diameter 15
Graph Density
HITS
PageRank
Connected Components
Community Detection
Modularity
Statistical Inference
Node Overview
Avg. Clustering Coefficient
Eigenvector Centrality
Edge Overview
Avg. Path Length 3.702
Dynamic
# Nodes
# Edges

Gephi is a powerful and flexible tool for visualizing and analyzing complex graphs and networks

It can read .dot file and generate above graph.

This website allows me to convert json nodes to Tree format. Easy to visualize and analyze complex graphs and networks. [Online JSON to Tree Diagram Converter (vanya.jp.net)](https://vanya.jp.net/vtree/)