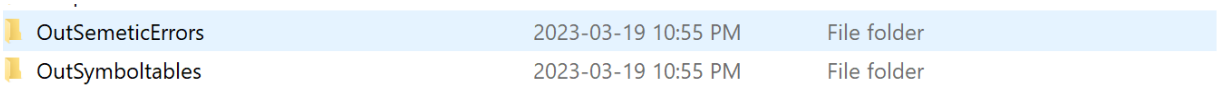
# Documentation4

Qiantongzhou40081938

Outputfile location:




## Section 1. List of semantic rules implemented

Visibility:

```
public Modifier modifier { get; set; }
```

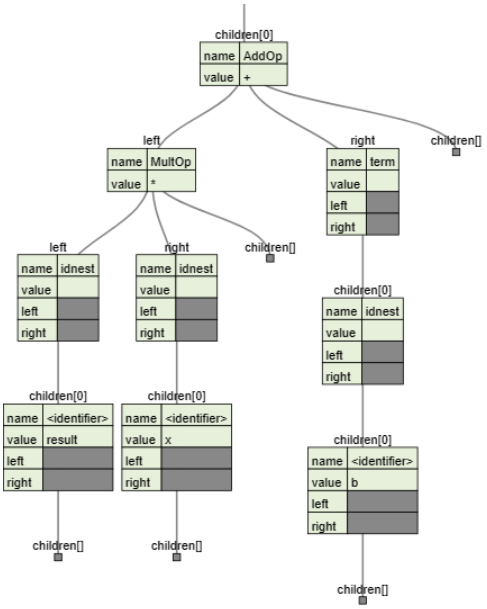
```
21 references
public enum Modifier
{
    Public,
    Private,
}
```

Modifier.Private = 1

Scope:

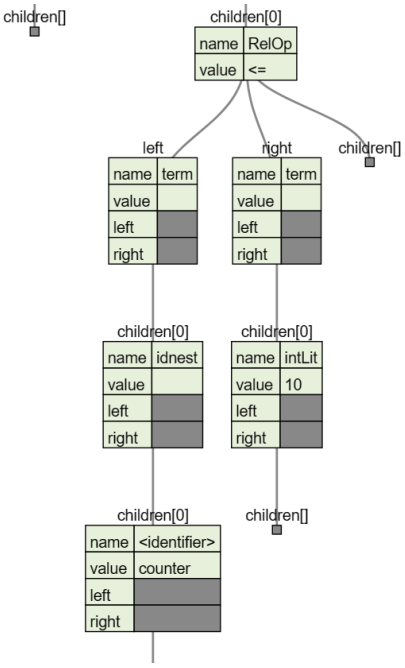
```
currentfunctiontable.currentScope++;
if (isfree)
{
    table.functiontables.Add(currentfunctiontable);
}
node.Children[2].Accept(this);
currentfunctiontable.currentScope--;
```

Mult:




Add:

Rel:




MembVarDecl:

```
2 references
public void Visit(memberVarDecl<T> node)
{
    var temp = new symbole(node.Children[0].Value, node.Children[3].Value, node.
    currentclasstable.symboltable.Add(temp);
}
```

MembFuncDecl:

```
2 references
public void Visit(memberfuncdecl<T> node)
{//member
    if (node.Children[0].Value == "function")
    {
        currentfunctiontable=new functiontable(node.Children[
        currentfunctiontable.functionparams=new List<symbole>
        for(int i = 4; i < node.Children.Count - 4; i++)
        {
            node.Children[i].Accept(this);
        }
        currentfunctiontable.returntype = new symbole("return

        currentclasstable.functiontables.Add(currentfunctiont

    }
```

FuncCall:

```
public void Visit(function<T> function)
{
    var temp=table.functiontables.FirstOrDefault(s => s.Name == function.Children[0]
    if (temp != null)
    {

    }
    else
    {errorlist.Add("cant find functioncall name:" + function.Children[0].Value);
        Console.WriteLine("cant find functioncall name:" + function.Children[0].Valu
    }
```

VarDecl:

```
2 references
public void Visit(LocalVarDecl<T> node)
{
    if (node.Name == "LocalVarDecl")
    {
        if (node.Children[4].Name == "Arraysize")
        {
            currentfunctiontable.symboltable.Add(new s
        }
        else
        {
            currentfunctiontable.symboltable.Add(new s
        }
    }
}
```

# Section 2. Design

The symbol table structure resolves scoping, binding, and typing of programme identifiers. This symbol table lists all identifiers (variables, functions, classes) in its scope. Each class definition, free or member function definition, and global programme scope have scopes.

Scope:

A programme has free functions, including one main function. Before binding and semantically checking free function calls, the symbol table must contain free function information. Implement at least two passes: a first pass that builds symbol tables and a second pass that uses that information to call functions before they are defined.

```
public abstract class table
{
    13 references
    public string Name { get; set; }
    public symboltable symboltable;
    public int currentScope = 0;
    3 references
    public table(string Name, symboltable symboltable)
    {
        this.Name = Name;
        this.symboltable = symboltable;
    }
}
```

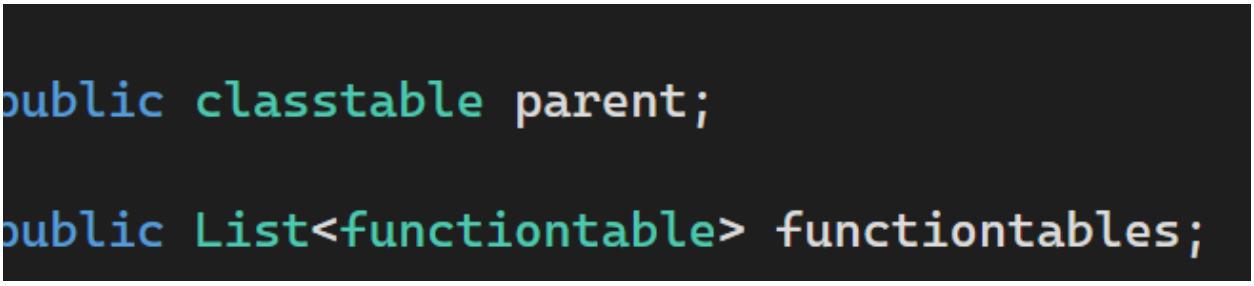
Classes:

Classes encapsulate user-defined data types and function declarations. Member functions are defined globally but use a scope-resolution operator to identify them as class members. To refer to a class declared after it, two passes are needed, just like free functions. The function declaration's symbol table entry must be bound to its local symbol table twice because member functions are declared in the class declaration and defined later.

```
10 references
public class classtable:table
{

    public classtable parent;

    public List<functiontable> functiontables;

    1 reference
    public classtable(string name, classtable parent=null, symboltabl
    {
        this.parent = parent;
        this.functiontables = functiontable;
    }
}
```

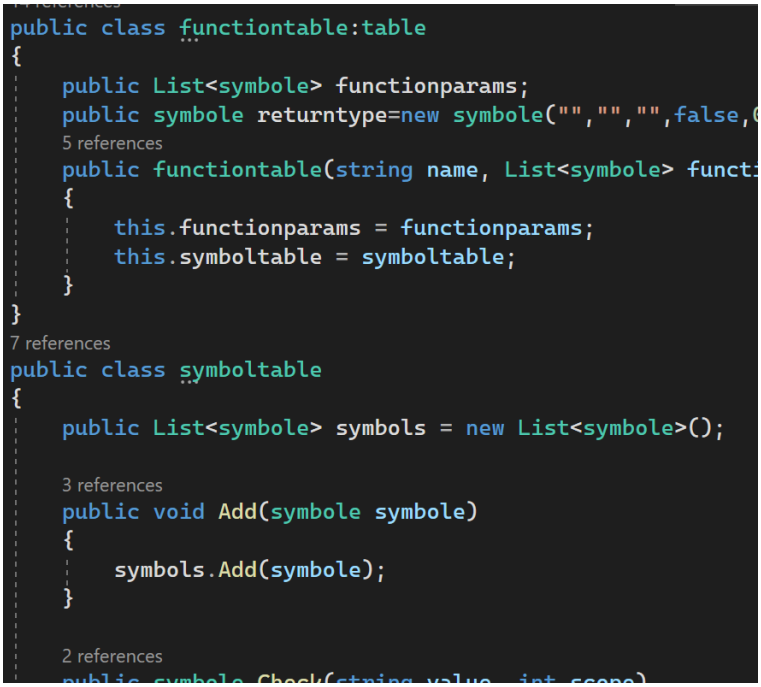
Inheritance:

If a class has an inheritance list, the symbol table of its directly inherited class(es) should be linked in this class to treat inherited members as class members even though they are in a different scope. Inherited members with the same name and type (variable or function) as a class member should be shadowed and warn. Circular class dependencies are always semantic errors.




Nested Symbol Tables:

Function and class variables are local and can only be used in the current function or class scope. All class member functions can use data members. This raises the need for a nested symbol table structure:



Attribute Migration:

When using operators in expressions, attribute migration must be done to determine the type of sub-expressions. For simplicity of code generation later, it is suggested that it should be semantically invalid to have operands of arithmetic operators be of different types. Assignment operands must also be the same type.

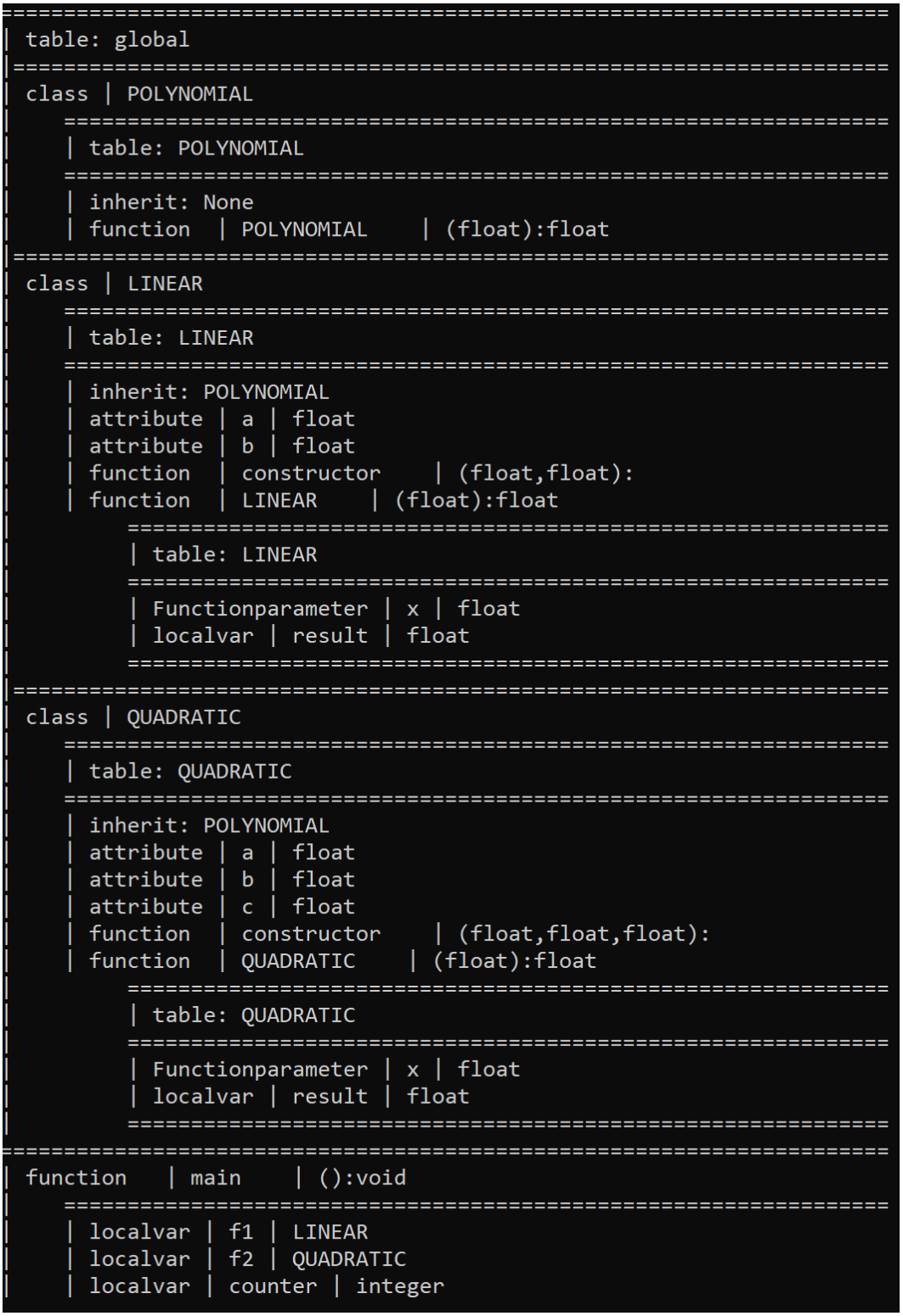
```
currentfunctiontable = new functiontable(node.Children[0].Value);
classtable link = table.classtables.FirstOrDefault(s => s.Name == node.Children[0].Valu
if (link != null)
{
    for (int i = 0; i < link.functiontables.Count; i++)
    {
        if (link.functiontables[i].Name == node.Children[2].Value)
        {
            var returntype = link.functiontables[i].returntype;
            link.functiontables[i] = currentfunctiontable;
            currentfunctiontable.returntype = returntype;
        }
    }

    node.Children[3].Accept(this);
```

Resulting table:

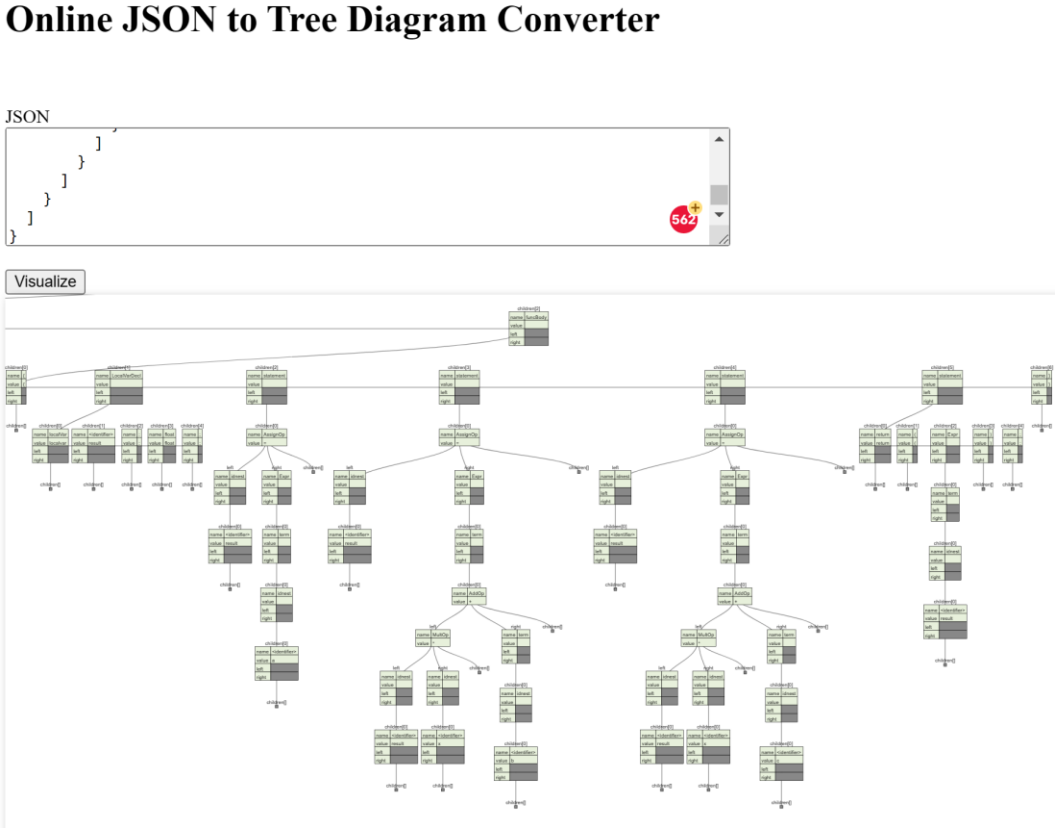
bubblesort

```
=============================================================================
| table: global
=============================================================================
| function   | bubbleSort    | (integer[],integer):void
|    ========================================================================
|    | localvar | n | integer
|    | localvar | i | integer
|    | localvar | j | integer
|    | localvar | temp | integer
|    ========================================================================
| function   | printarray    | (integer[],integer):void
|    ========================================================================
|    | localvar | n | integer
|    | localvar | i | integer
|    ========================================================================
| function   | main    | ():void
|    ========================================================================
|    | localvar | arr | integer
|    ========================================================================
```

Inheritance and linked functions



# Section 3. Use of tools




[Online JSON to Tree Diagram Converter (vanya.jp.net)]

Visitor pattern