# Assignment 4 Semantic Analysis

## Section 1 List of semantic rules implemented

1. global symbol table
2. class symbol table

3.1 class data member
3.2 function's local variable
4.1 free function
4.2 member function

6.1 undeclared member function definition
6.2 undefined member function declaration

8.1 multiply declared class
8.2 multiply defined free function
8.3 multiply declared identifier in class
8.4 multiply declared identifier in function
8.5 shadowed inherited data member
8.6 multiply declared function in class

9.1 Overloaded free function
9.2 Overloaded member function
9.3 Overridden inherited member function

10.1 Type error in expression
10.2 Type error in assignment statement
10.3 Type error in return statement

11.1 Undeclared local variable
11.2 Undeclared data member
11.2.1 Undeclared variable (if function is member function, check in the class for member used as variable)
11.2.2 Undeclared variable (if function is member function, and its class inherits search in all super classes' tables for member used as variable)
11.2.3 Undeclared data member (search in class table)
11.2.4 Undeclared data member (if class inherits from other classes, search in all superclasses' tables)

11.3 Undeclared member function
11.3.1 Undeclared member function (search in class table)
11.3.2 Undeclared member function (if class inherits from other classes, search in all superclasses' tables)

11.4 Undeclared free function
11.5 Undeclared class

12.1 Function call with wrong number of parameters
12.2 Function call with wrong type of parameters

13.1 Use of array with wrong number of dimensions
13.2 Array index is not an integer
13.3 Array parameter using wrong number of dimensions

14.1 Circular class dependency

15.1 "." operator used on non-class type

All semantic rules tests can be found in SymbolTableSemanticCheckingTest.java, which runs corresponding test .src and checks the results.

## Section 2 Design

### Overall design

Package *symboltable* contains the symbol table data structure and symbol table entry data structure. SymbolTable.java defines the symbol table structure, which contains a list of entries defined as SymbolTableEntry.java. Each symbol table entry links to another symbol table if applied. Also, the symbol table has reference to the parent symbol table if applied.

Package *visitor* contains the concrete visitor pattern implementation for symbol table creation and semantic rules checking. Two concrete visitors are SybolTableCreationVisitor.java and SemanticCheckingVisitor.java

### Phase

Symbol table creation
This phase is done right after AST generation, which constructs the symbol table by traversing each node in AST. SymbolTableCreationVisitor.java implements the logic for this phase. Symbol tables are created for AST node prog, funcDef, structDecl. Each symbol table also has an entry in the parent symbol table and has a link to the parent table. Symbol table is passed to the AST children node, so that the parameters, variables and functions defined in scope are added in the symbol table as well.

Link function in impl to the function in struct
In given grammar, function declaration and definition is splitted between struct and impl. Therefore a helper class Uilt.java is implemented and called after symbol table creation to link

the function declared in struct with the implementation in impl. This class also brings inherited members in parent struct to the child struct if members are not overwritten. Some semantic checks, such as circular class dependency, undefined member function declaration, undeclared member function definition, etc, are implemented in this phase as well.

Semantic rules checking
This is the third phase for semantic analysis. SemanticCheckingVisitor.java is implemented to traverse the AST and use the symbol table created earlier to perform different semantic rules validation. The AST node that implements the semantic checking are addOp, multOp, var, varDecl, fCall, sign, dot, returnStat, assignStat

## Section 3 Use of tools

No new tools used in the assignment

### Assignment Submission

/src - contains the source code

/input/src - contains the test files

/output - contains the output files, separated by folder with the test file name as folder name

/semanticanalyzerdriver.jar - executable

Jar executable is built by Maven using Java11, which requires Java Runtime Environment 11+
If you wish to rebuild the jar, use the pom.xml. Otherwise, you can compile and run the program from source code.

#### How to run jar executable

Run with default .src files in /input folder

```
 $ java -jar semanticanalyzerdriver.jar ./input/src/bubblesort.src
```

output files will be available in ./output/bubblesort/