Compilers – II: Semantic Analysis

Team 4:

Aman Panwar - CS20BTECH11004	Shreya Kumar - ES20BTECH11026
Pranav K Nayak - ES20BTECH11035	Taha Adeel Mohammed - CS20BTECH11052
Shambhu Kavir - CS20BTECH11045	Vikhyath - CS20BTECH11056

Goals of Assignment

- Generating an AST for our language
- Performing semantic analysis for our code including
  - Generating the symbol table,
  - Performing type checking and checking of scopes.

Creating the AST

- We first created the Class Hierarchy for our language. The header file **astNodes.h** consists of this Class Hierarchy implemented as C++ classes and uses Inheritance.
- The implementation of the member functions and constructors of all the classes of the hierarchy are present in the file **astNodes.cpp.**
- We have started **integrating the AST with the parser.** This is done by creating new AST Nodes in the actions of the grammar and connecting them using pointers.

Symbol Table

- We use a linked list to store all the different symbols.
- Each symbol holds attributes including the file it is present in and all the instances at which it is referenced.
- We have created routines to store the symbols along with their attributes, and routines to print them.

Challenges Faced

- We struggled with creating the AST using the grammar actions.

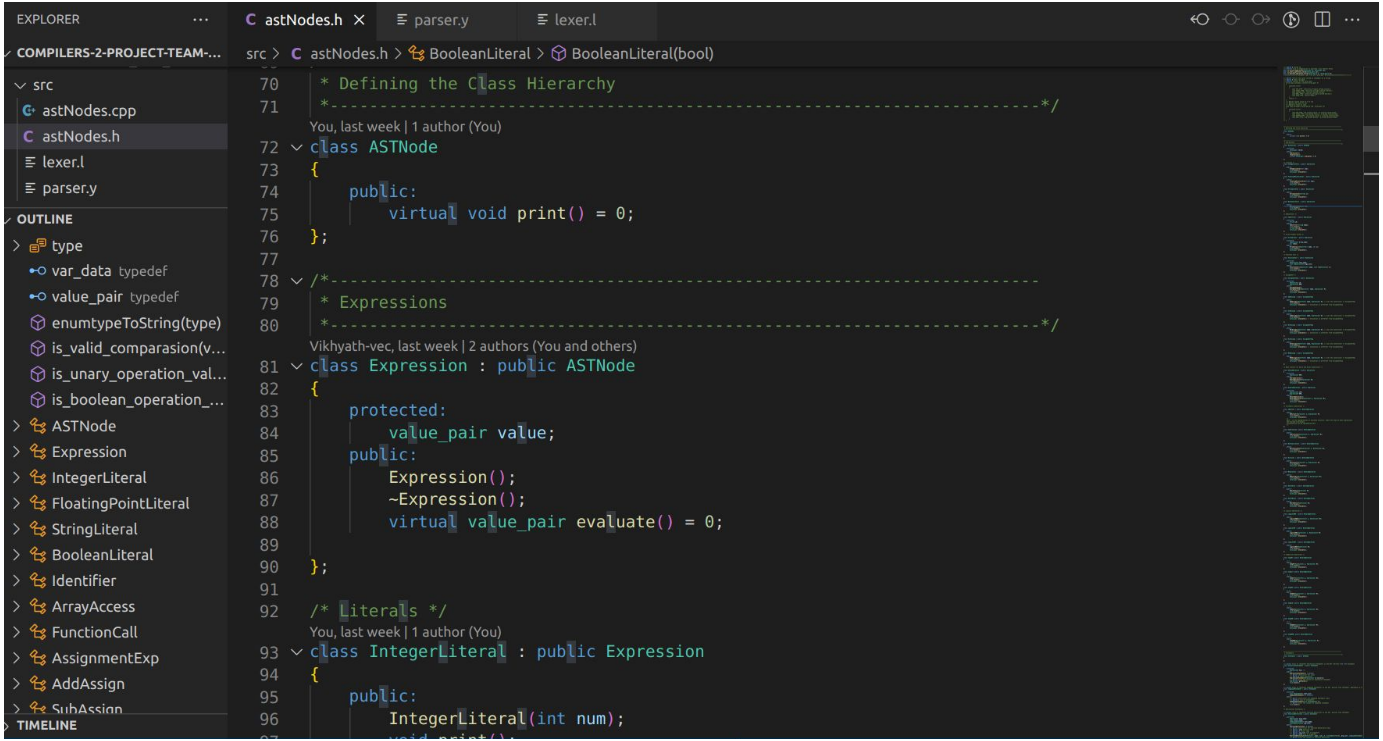
  In practice we ended up reiterating the code a few times as went on and recognised our mistakes and possible improvements

- The data checking required us to manually write code for every type of variable with every other type of variable and for every operation. This proved to be cumbersome and difficult to debug.

  We used std::variant instead of union to simplify our task.

Next Steps...

- Integrating the AST into the parser, by creating nodes using the grammar actions.
- Traversing the AST using the symbol table to deal with scope and type checking.
- Making our AST compatible with LLVM to start with the Code Generation phase.



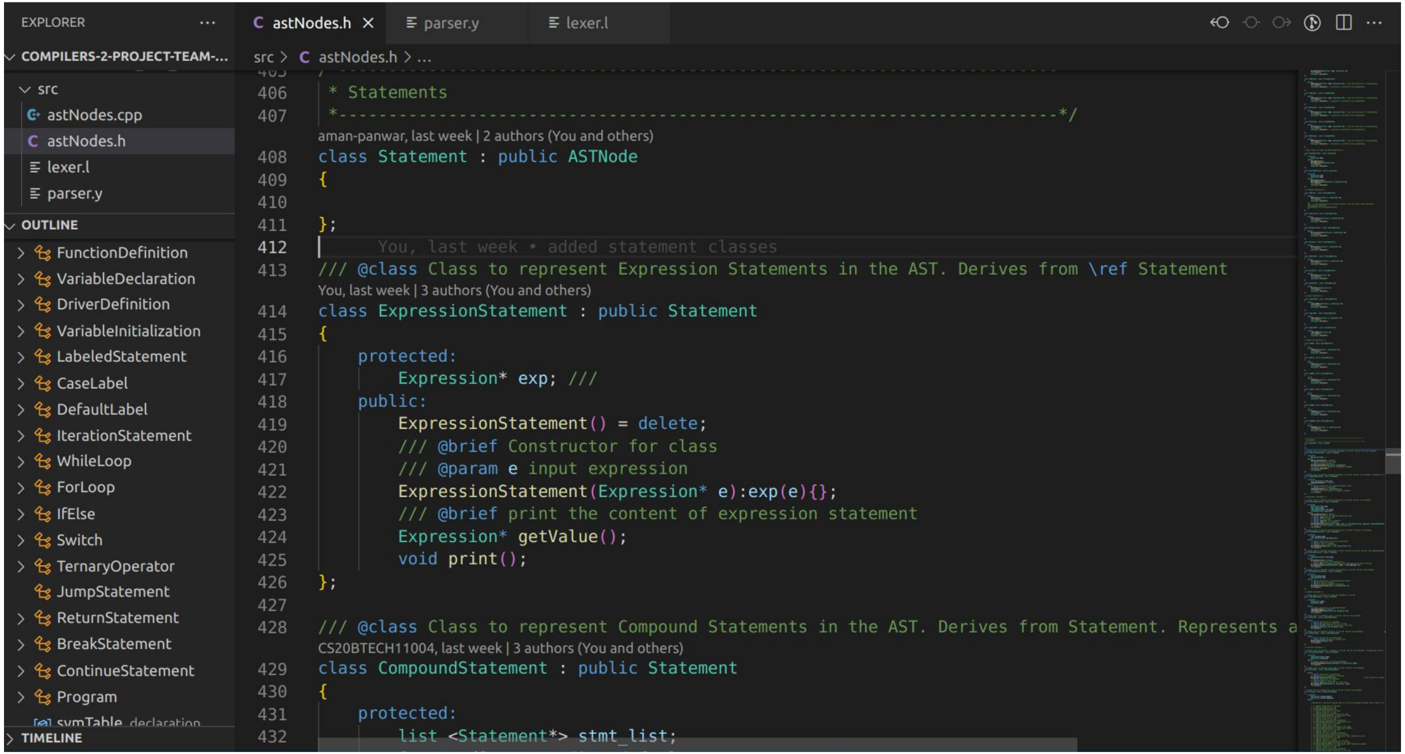
```
 * Defining the Class Hierarchy
 *------------------------------------------------------------------------*/
You, last week | 1 author (You)
class ASTNode
{
    public:
        virtual void print() = 0;
};

/*------------------------------------------------------------------------
 * Expressions
 *------------------------------------------------------------------------*/
Vikhyath-vec, last week | 2 authors (You and others)
class Expression : public ASTNode
{
    protected:
        value_pair value;
    public:
        Expression();
        ~Expression();
        virtual value_pair evaluate() = 0;

};

/* Literals */
You, last week | 1 author (You)
class IntegerLiteral : public Expression
{
    public:
        IntegerLiteral(int num);
```

CODE SNIPPETS: A Glimpse of our code hierarchy



```
 * Statements
 *---------------------------------------------------------------------------*/
aman-panwar, last week | 2 authors (You and others)
class Statement : public ASTNode
{

};
        You, last week • added statement classes
/// @class Class to represent Expression Statements in the AST. Derives from \ref Statement
You, last week | 3 authors (You and others)
class ExpressionStatement : public Statement
{
    protected:
        Expression* exp; ///
    public:
        ExpressionStatement() = delete;
        /// @brief Constructor for class
        /// @param e input expression
        ExpressionStatement(Expression* e):exp(e){};
        /// @brief print the content of expression statement
        Expression* getValue();
        void print();
};

/// @class Class to represent Compound Statements in the AST. Derives from Statement. Represents a
CS20BTECH11004, last week | 3 authors (You and others)
class CompoundStatement : public Statement
{
    protected:
        list <Statement*> stmt_list;
```

CODE SNIPPETS: A Glimpse of our code hierarchy