



Compilers – II: Semantic Analysis

Team 4:

Aman Panwar - CS20BTECH11004

Pranav K Nayak - ES20BTECH11035

Shambhu Kavir - CS20BTECH11045

Shreya Kumar - ES20BTECH11026

Taha Adeel Mohammed - CS20BTECH11052

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 we went on and recognised our mistakes and possible improvements

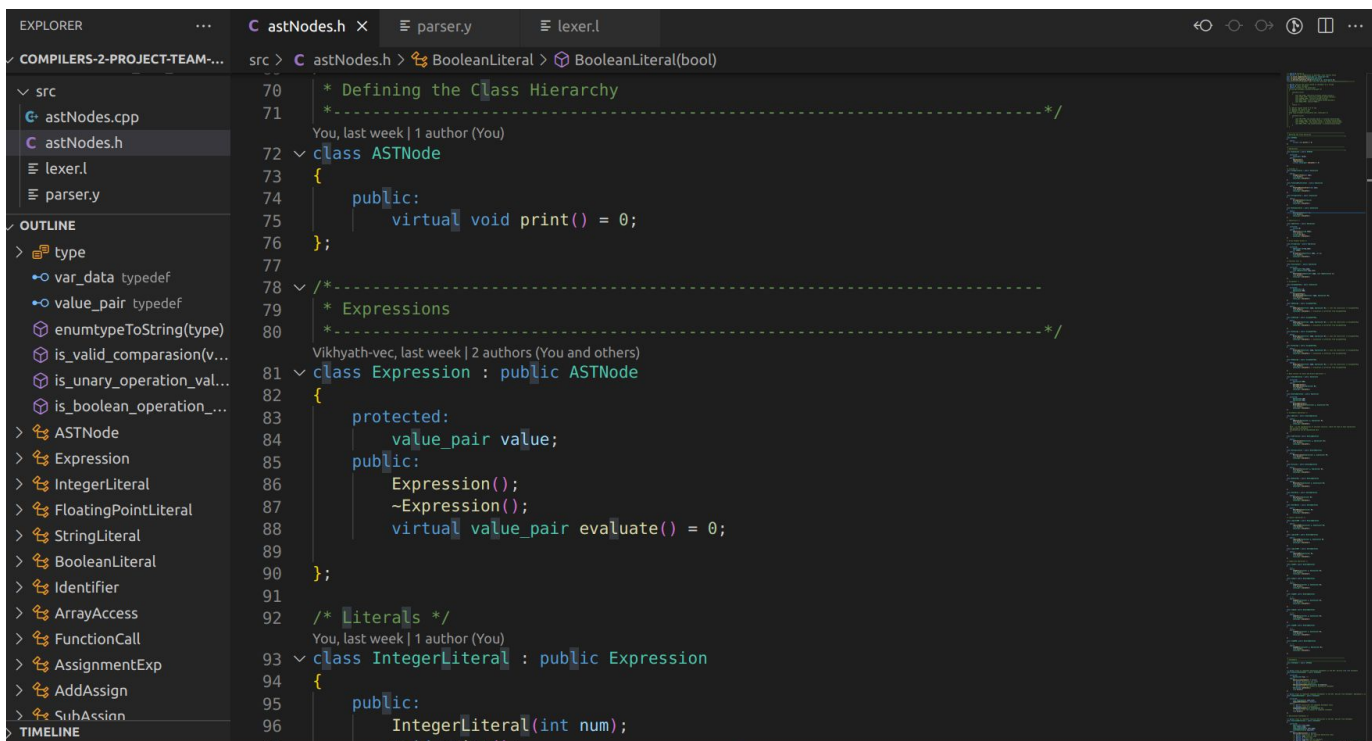
- The data checking required us to manually write code for every type of variable with every other type of variable and for every operation. This proved to be cumbersome and difficult to debug.

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.



```
70  * Defining the Class Hierarchy
71  *-----*/
72  You, last week | 1 author (You)
73  class ASTNode
74  {
75  public:
76      virtual void print() = 0;
77  };
78  *-----*/
79  * Expressions
80  *-----*/
81  Vikhyath-vec, last week | 2 authors (You and others)
82  class Expression : public ASTNode
83  {
84  protected:
85      value_pair value;
86  public:
87      Expression();
88      ~Expression();
89      virtual value_pair evaluate() = 0;
90  };
91
92  /* Literals */
93  You, last week | 1 author (You)
94  class IntegerLiteral : public Expression
95  {
96  public:
97      IntegerLiteral(int num);
98      void print();
```

The image shows a code editor with a dark theme. On the left, the 'EXPLORER' sidebar shows a project named 'COMPILERS-2-PROJECT-TEAM-...' with a 'src' folder containing 'astNodes.cpp', 'astNodes.h', 'lexer.l', and 'parser.y'. Below this is an 'OUTLINE' view showing a tree structure of the code: 'type' (with 'var_data typedef' and 'value_pair typedef'), 'enumtypeToString(type)', 'is_valid_comparasion(v...', 'is_unary_operation_val...', 'is_boolean_operation_...', 'ASTNode', 'Expression', 'IntegerLiteral', 'FloatingPointLiteral', 'StringLiteral', 'BooleanLiteral', 'Identifier', 'ArrayAccess', 'FunctionCall', 'AssignmentExp', 'AddAssign', and 'SubAssign'. The main editor area shows the 'astNodes.h' file. It contains a class hierarchy for an abstract syntax tree. The 'ASTNode' class is the base class, with a 'virtual void print() = 0;' method. The 'Expression' class inherits from 'ASTNode' and has a 'protected: value_pair value;' member, a 'virtual value_pair evaluate() = 0;' method, and a 'void print();' method. The 'IntegerLiteral' class inherits from 'Expression' and has a 'void print();' method. The code is commented with '/* Defining the Class Hierarchy */' and '/* Expressions */'. The editor also shows a 'TIMELINE' view on the right side.

CODE SNIPPETS: A Glimpse of our code hierarchy

The screenshot shows a code editor with a dark theme. On the left, the 'EXPLORER' sidebar displays a project structure for 'COMPILERS-2-PROJECT-TEAM-...'. It includes a 'src' folder containing 'astNodes.cpp', 'astNodes.h', 'lexer.l', and 'parser.y'. Below this is an 'OUTLINE' section listing various code snippets: FunctionDefinition, VariableDeclaration, DriverDefinition, VariableInitialization, LabeledStatement, CaseLabel, DefaultLabel, IterationStatement, WhileLoop, ForLoop, IfElse, Switch, TernaryOperator, JumpStatement, ReturnStatement, BreakStatement, ContinueStatement, and Program. A 'TIMELINE' section at the bottom shows a snippet titled 'sumTable_declaration'. The main editor area displays the 'astNodes.h' file, showing C++ code for the AST classes. The code includes comments and class definitions for 'Statement', 'ExpressionStatement', and 'CompoundStatement'. The 'Statement' class is the base class, and 'ExpressionStatement' and 'CompoundStatement' inherit from it. The 'CompoundStatement' class has a protected member 'list <Statement*> stmt_list;'. The code is written in a C++ style with comments in green and code in white/yellow.

```
405
406 * Statements
407 *-----*/
408 aman-panwar, last week | 2 authors (You and others)
409 class Statement : public ASTNode
410 {
411 };
412 | You, last week * added statement classes
413 /// @class Class to represent Expression Statements in the AST. Derives from \ref Statement
414 You, last week | 3 authors (You and others)
415 class ExpressionStatement : public Statement
416 {
417 protected:
418     Expression* exp; ///
419 public:
420     ExpressionStatement() = delete;
421     /// @brief Constructor for class
422     /// @param e input expression
423     ExpressionStatement(Expression* e):exp(e){};
424     /// @brief print the content of expression statement
425     Expression* getValue();
426     void print();
427 };
428 /// @class Class to represent Compound Statements in the AST. Derives from Statement. Represents a
429 CS20BTECH11004, last week | 3 authors (You and others)
430 class CompoundStatement : public Statement
431 {
432 protected:
433     list <Statement*> stmt_list;
```

CODE SNIPPETS: A Glimpse of our code hierarchy