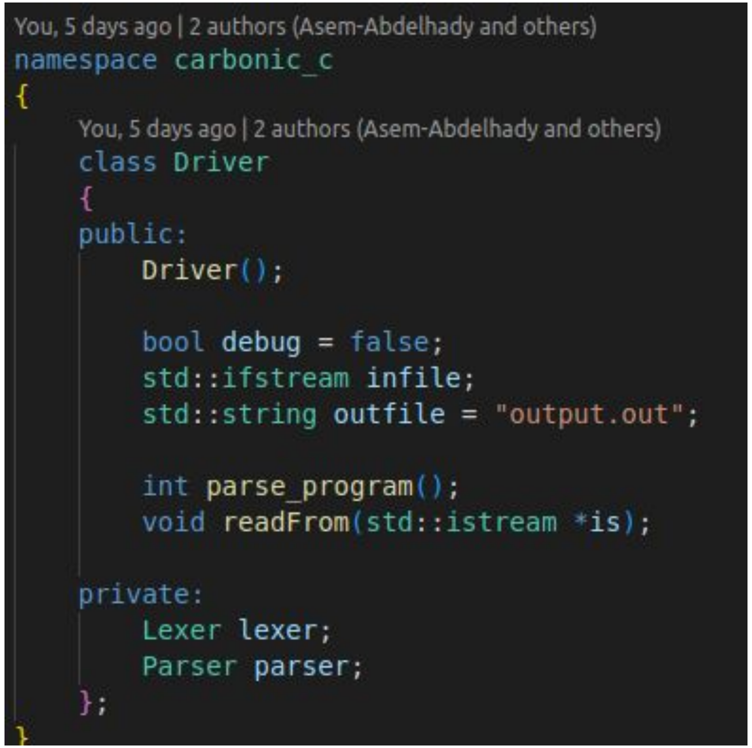
Carbonic-C
Syntax analyzer

Asem, Jaafar, Mosab, Menna

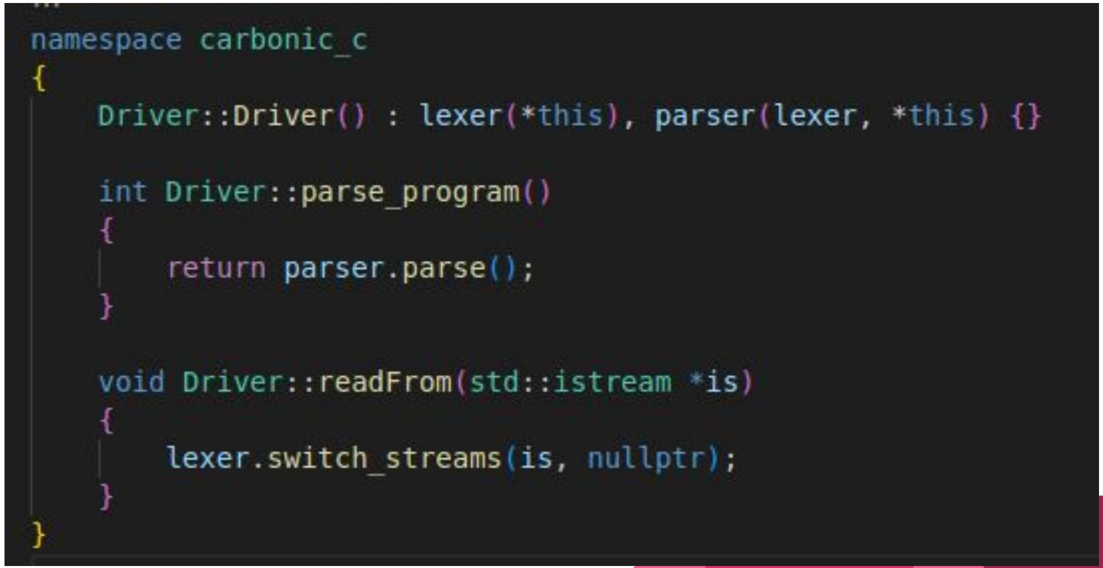
Parser implementation - Linking Lexer and Parser

- Driver class as a composer.
- Redefined Parser symbol type.
- Matching Parser and Lexer tokens and literals.
- Parser has types for literals.

Parser implementation - Linking Lexer and Parser



```
You, 5 days ago | 2 authors (Asem-Abdelhady and others)
namespace carbonic_c
{
    You, 5 days ago | 2 authors (Asem-Abdelhady and others)
    class Driver
    {
    public:
        Driver();

        bool debug = false;
        std::ifstream infile;
        std::string outfile = "output.out";

        int parse_program();
        void readFrom(std::istream *is);

    private:
        Lexer lexer;
        Parser parser;
    };
}
```

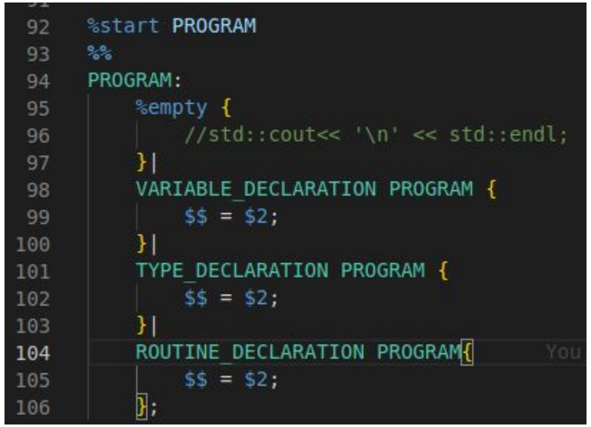


```
namespace carbonic_c
{
    Driver::Driver() : lexer(*this), parser(lexer, *this) {}

    int Driver::parse_program()
    {
        return parser.parse();
    }

    void Driver::readFrom(std::istream *is)
    {
        lexer.switch_streams(is, nullptr);
    }
}
```

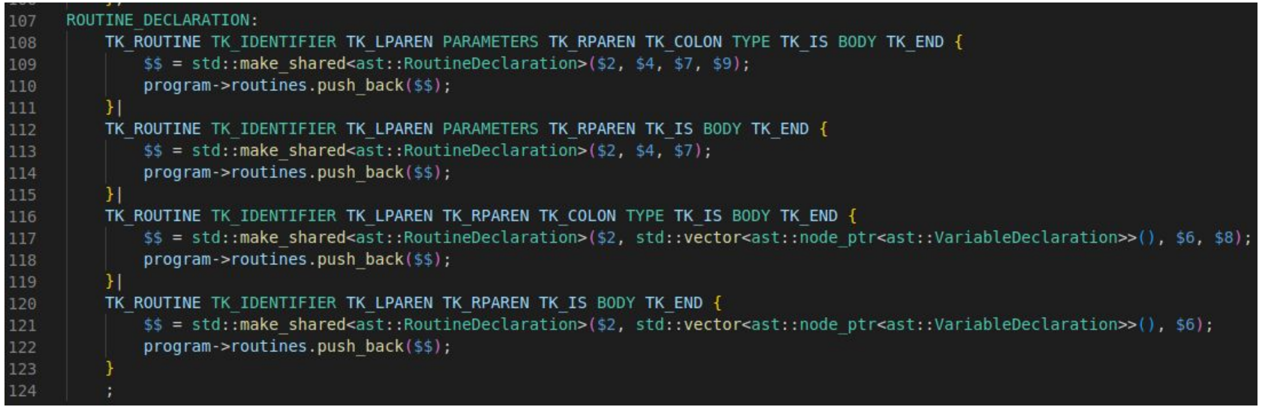
Parser implementation - Implementing Parser

- Defining grammar.
- Returning AST nodes.
- Initialize AST program.

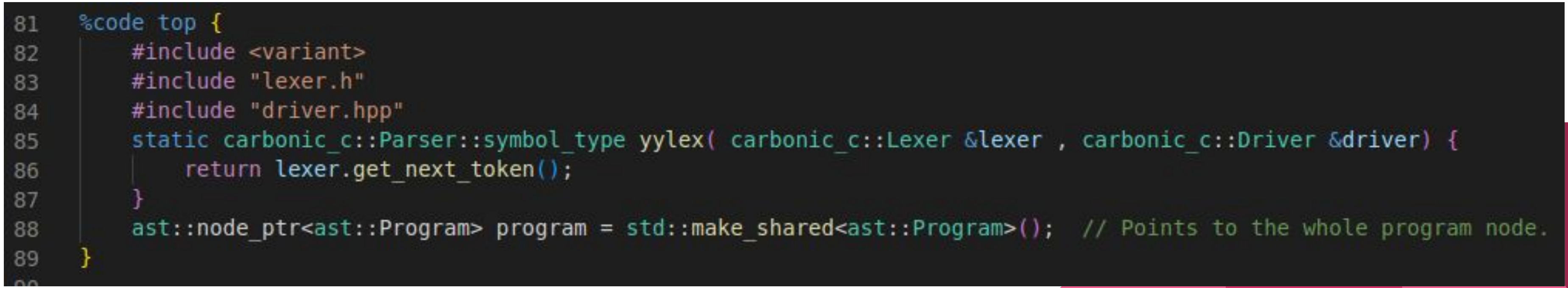
Parser implementation - Implementing Parser



```
%start PROGRAM
%%
PROGRAM:
    %empty {
        //std::cout<< '\n' << std::endl;
    }|
    VARIABLE_DECLARATION PROGRAM {
        $$ = $2;
    }|
    TYPE_DECLARATION PROGRAM {
        $$ = $2;
    }|
    ROUTINE_DECLARATION PROGRAM{
        $$ = $2;
    };
```



```
ROUTINE_DECLARATION:
    TK_ROUTINE TK_IDENTIFIER TK_LPAREN PARAMETERS TK_RPAREN TK_COLON TYPE TK_IS BODY TK_END {
        $$ = std::make_shared<ast::RoutineDeclaration>($2, $4, $7, $9);
        program->routines.push_back($$);
    }|
    TK_ROUTINE TK_IDENTIFIER TK_LPAREN PARAMETERS TK_RPAREN TK_IS BODY TK_END {
        $$ = std::make_shared<ast::RoutineDeclaration>($2, $4, $7);
        program->routines.push_back($$);
    }|
    TK_ROUTINE TK_IDENTIFIER TK_LPAREN TK_RPAREN TK_COLON TYPE TK_IS BODY TK_END {
        $$ = std::make_shared<ast::RoutineDeclaration>($2, std::vector<ast::node_ptr<ast::VariableDeclaration>>(), $6, $8);
        program->routines.push_back($$);
    }|
    TK_ROUTINE TK_IDENTIFIER TK_LPAREN TK_RPAREN TK_IS BODY TK_END {
        $$ = std::make_shared<ast::RoutineDeclaration>($2, std::vector<ast::node_ptr<ast::VariableDeclaration>>(), $6);
        program->routines.push_back($$);
    }
    ;
```

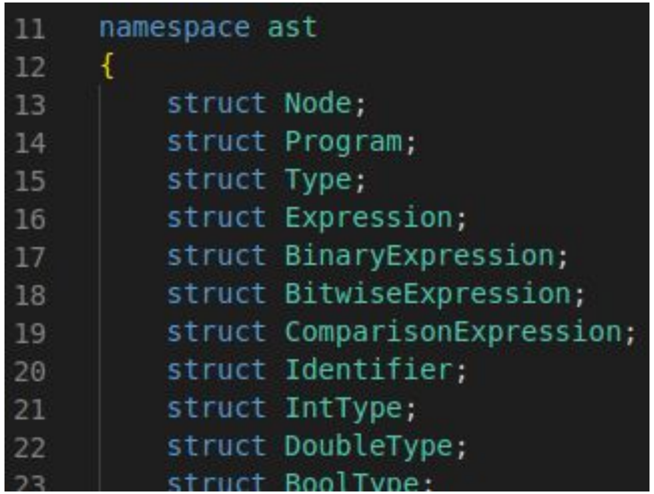


```
%code top {
    #include <variant>
    #include "lexer.h"
    #include "driver.hpp"
    static carbonic_c::Parser::symbol_type yylex( carbonic_c::Lexer &lexer , carbonic_c::Driver &driver) {
        return lexer.get_next_token();
    }
    ast::node_ptr<ast::Program> program = std::make_shared<ast::Program>();  // Points to the whole program node.
}
```

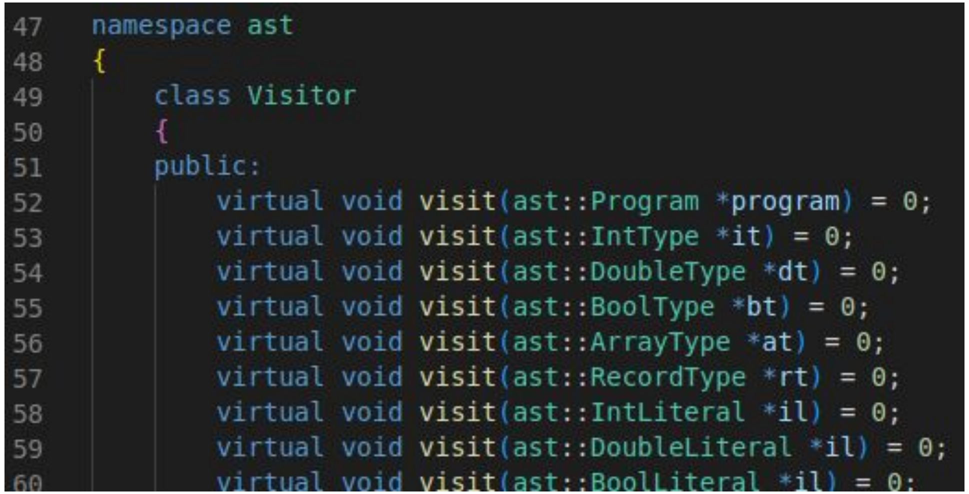
Parser implementation - AST

- Create AST Class and namespace.
- Shared pointer for all nodes (node_ptr).
- Visitor design pattern.
- Visitor abstract class and virtual visits.
- Visitable node structs.
- Nodes as structs.

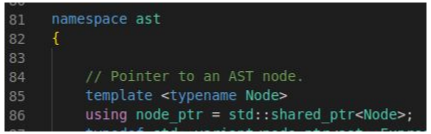
Parser implementation - AST



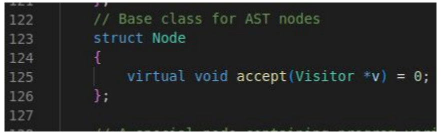
```cpp
namespace ast
{
    struct Node;
    struct Program;
    struct Type;
    struct Expression;
    struct BinaryExpression;
    struct BitwiseExpression;
    struct ComparisonExpression;
    struct Identifier;
    struct IntType;
    struct DoubleType;
    struct BoolType;
```



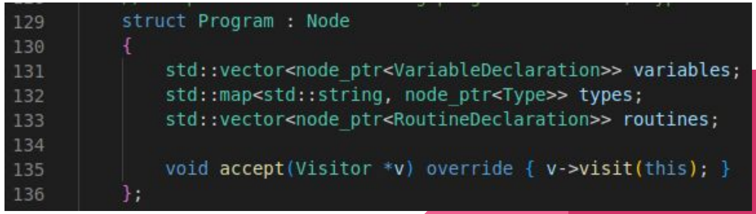
```cpp
namespace ast
{
    class Visitor
    {
    public:
        virtual void visit(ast::Program *program) = 0;
        virtual void visit(ast::IntType *it) = 0;
        virtual void visit(ast::DoubleType *dt) = 0;
        virtual void visit(ast::BoolType *bt) = 0;
        virtual void visit(ast::ArrayType *at) = 0;
        virtual void visit(ast::RecordType *rt) = 0;
        virtual void visit(ast::IntLiteral *il) = 0;
        virtual void visit(ast::DoubleLiteral *il) = 0;
        virtual void visit(ast::BoolLiteral *il) = 0;
```



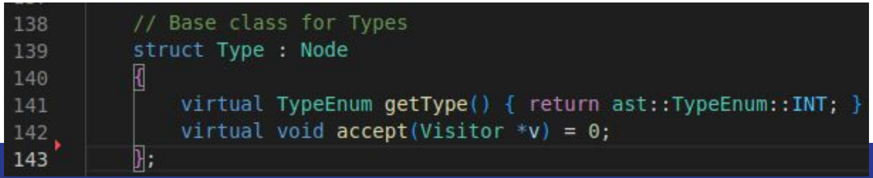
```cpp
namespace ast
{

    // Pointer to an AST node.
    template <typename Node>
    using node_ptr = std::shared_ptr<Node>;
```



```cpp
    // Base class for AST nodes
    struct Node
    {
        virtual void accept(Visitor *v) = 0;
    };

```



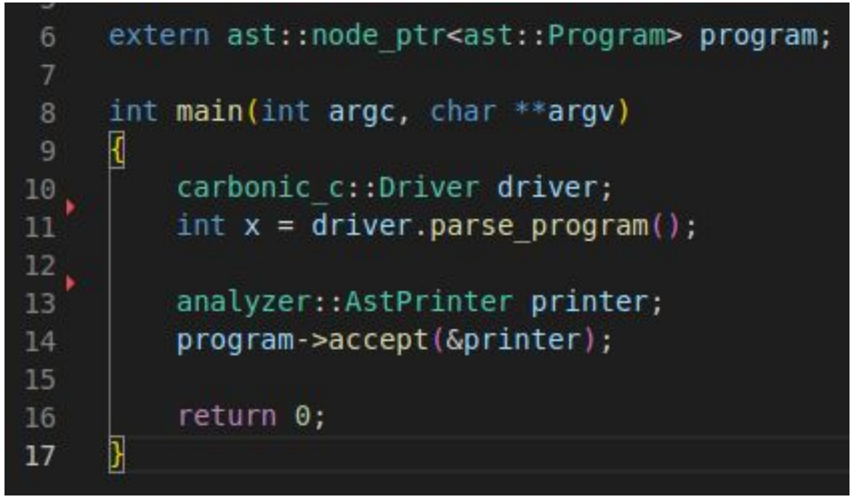
```cpp
    struct Program : Node
    {
        std::vector<node_ptr<VariableDeclaration>> variables;
        std::map<std::string, node_ptr<Type>> types;
        std::vector<node_ptr<RoutineDeclaration>> routines;

        void accept(Visitor *v) override { v->visit(this); }
    };
```



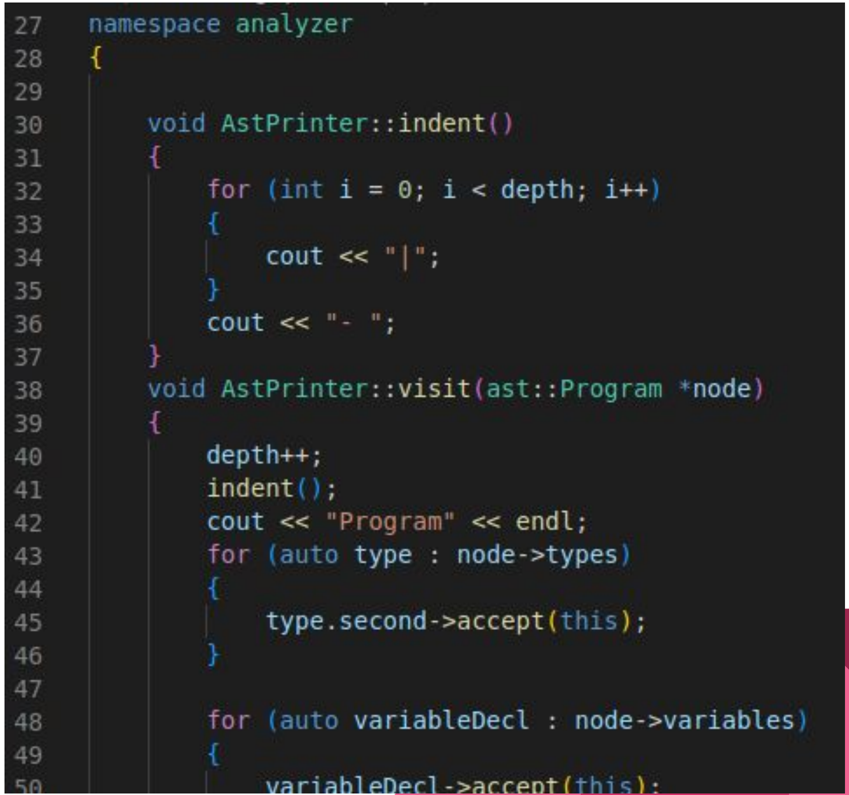
```cpp
    // Base class for Types
    struct Type : Node
    {
        virtual TypeEnum getType() { return ast::TypeEnum::INT; }
        virtual void accept(Visitor *v) = 0;
    };
```

Parser implementation - Visualization

- Using the visitor for printing

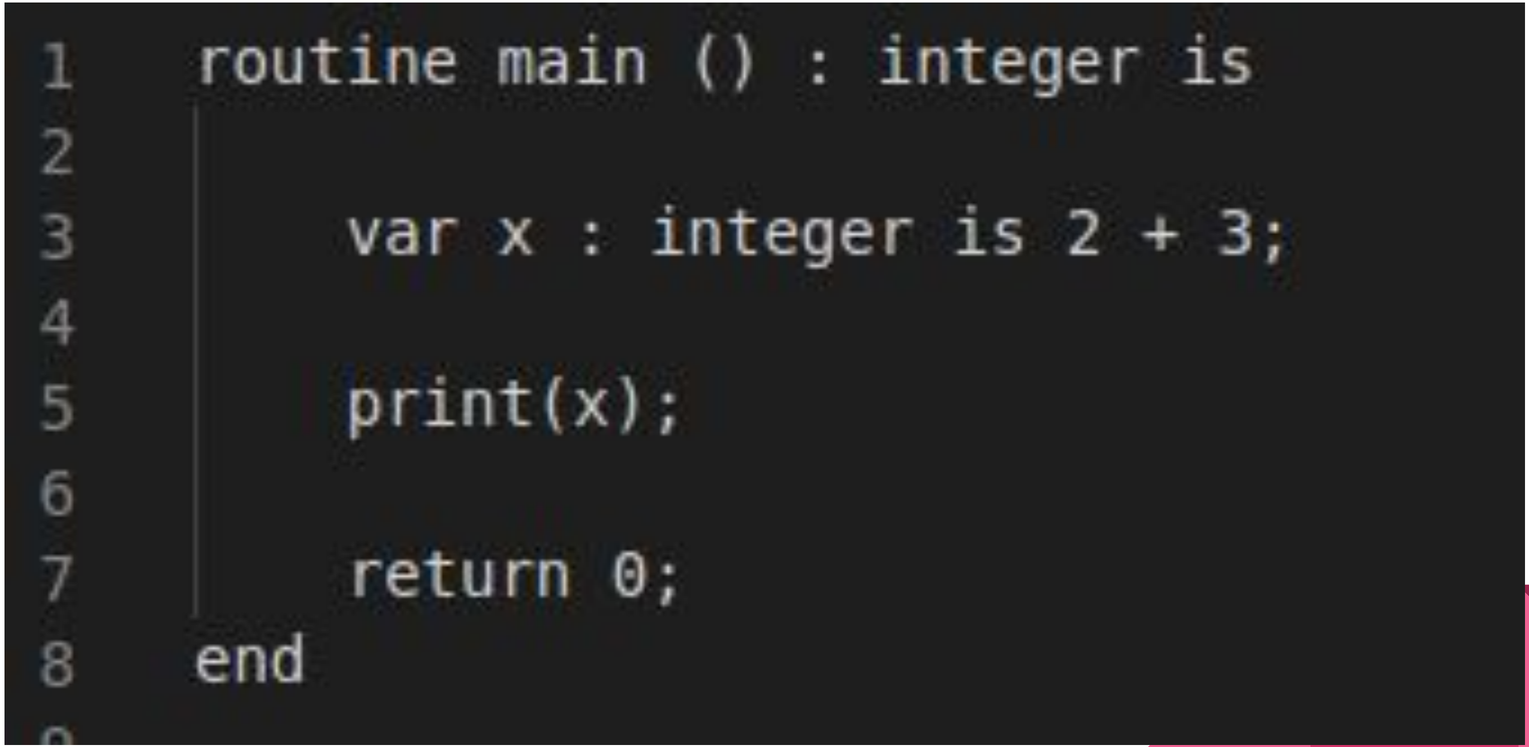


```
extern ast::node_ptr<ast::Program> program;

int main(int argc, char **argv)
{
    carbonic_c::Driver driver;
    int x = driver.parse_program();

    analyzer::AstPrinter printer;
    program->accept(&printer);

    return 0;
}
```



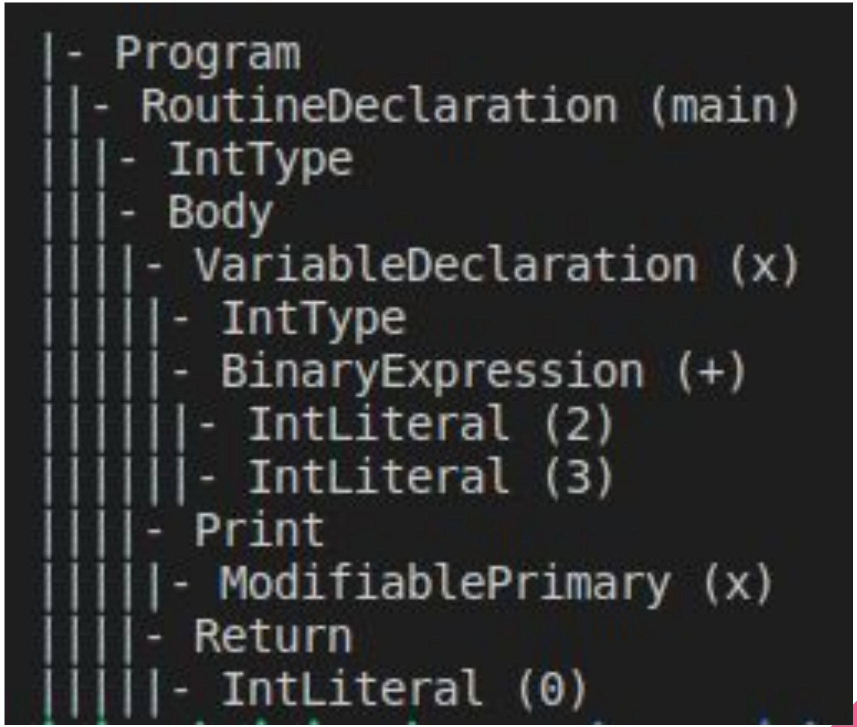
```
namespace analyzer
{

    void AstPrinter::indent()
    {
        for (int i = 0; i < depth; i++)
        {
            cout << "|";
        }
        cout << "- ";
    }
    void AstPrinter::visit(ast::Program *node)
    {
        depth++;
        indent();
        cout << "Program" << endl;
        for (auto type : node->types)
        {
            type.second->accept(this);
        }

        for (auto variableDecl : node->variables)
        {
            variableDecl->accept(this);
```

Example #1 (Simple)



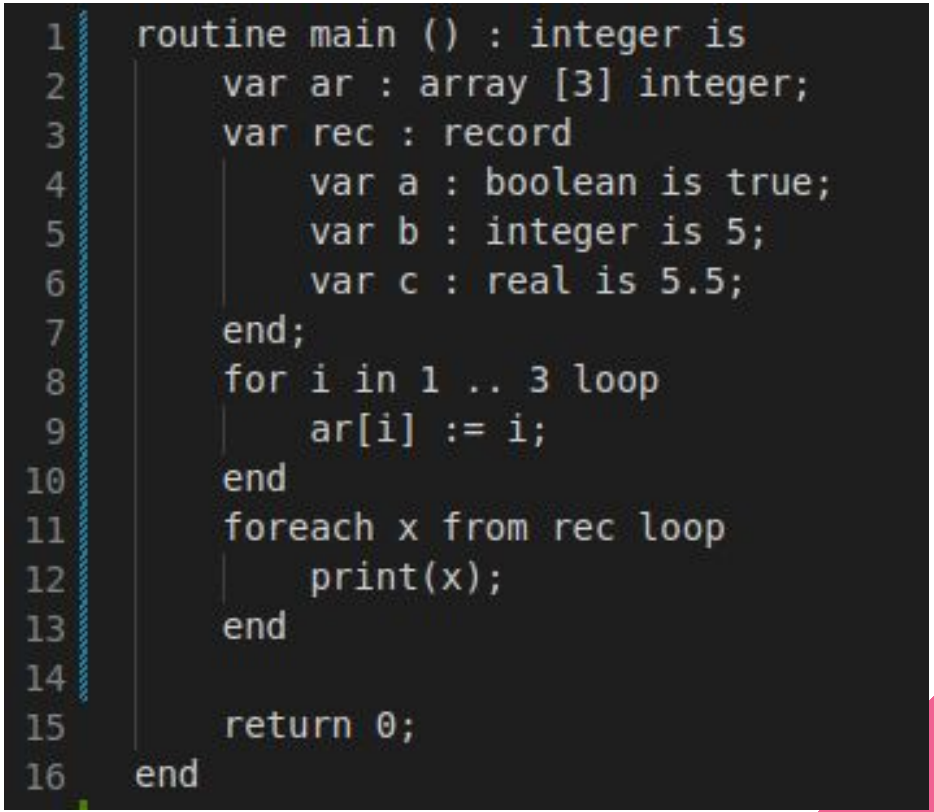
```
routine main () : integer is

    var x : integer is 2 + 3;

    print(x);

    return 0;
end
```

Example #1 - AST Visualization



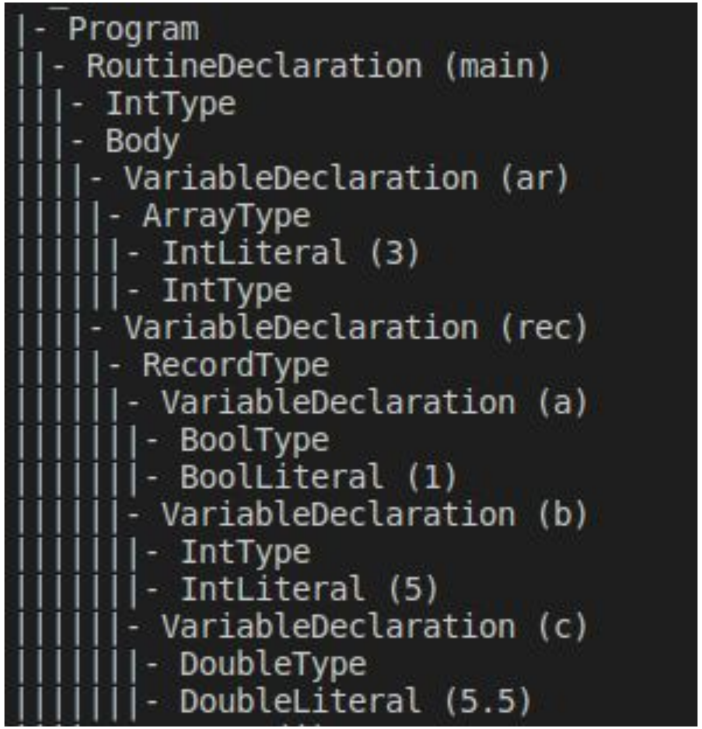
```
|- Program
||- RoutineDeclaration (main)
|||- IntType
|||- Body
||||- VariableDeclaration (x)
|||||- IntType
|||||- BinaryExpression (+)
||||||- IntLiteral (2)
||||||- IntLiteral (3)
||||- Print
|||||- ModifiablePrimary (x)
||||- Return
|||||- IntLiteral (0)
```

Example #2 (Intermediate)

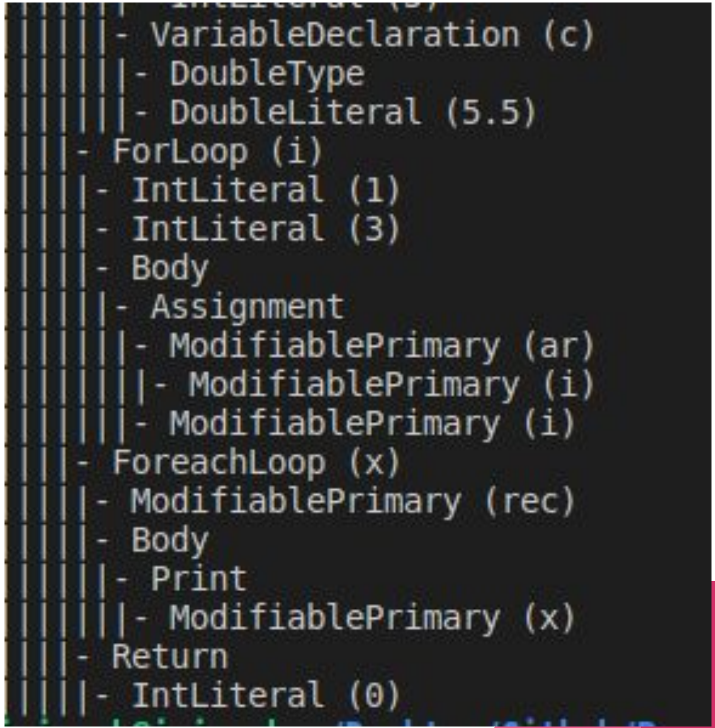


```
routine main () : integer is
    var ar : array [3] integer;
    var rec : record
        var a : boolean is true;
        var b : integer is 5;
        var c : real is 5.5;
    end;
    for i in 1 .. 3 loop
        ar[i] := i;
    end
    foreach x from rec loop
        print(x);
    end

    return 0;
end
```

Example #2 - AST Visualization

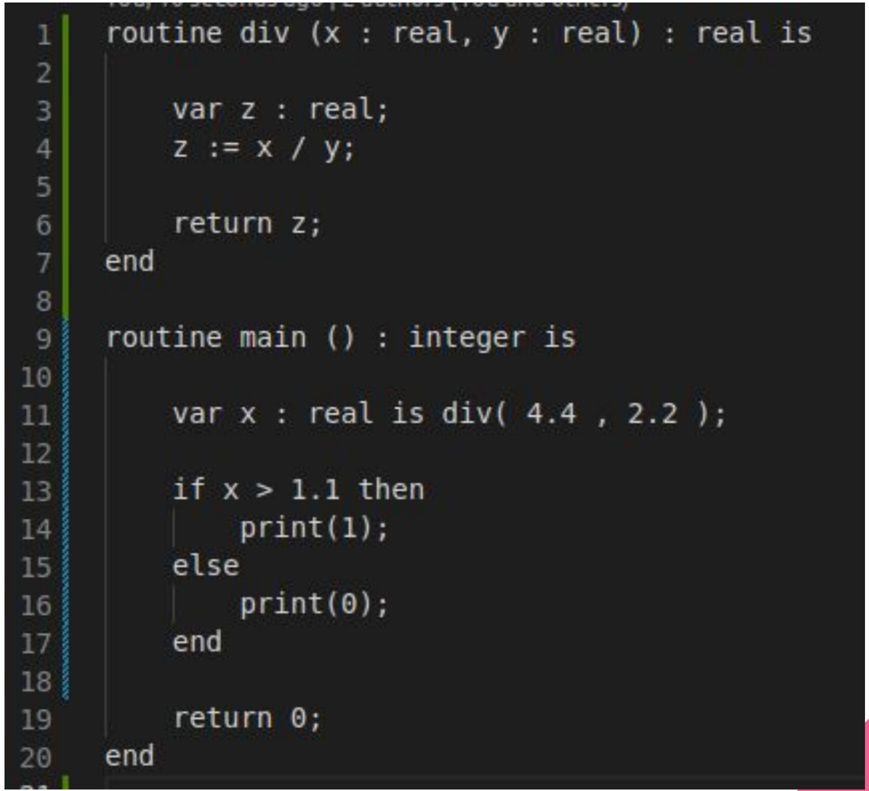


```
|- Program
||- RoutineDeclaration (main)
|||- IntType
|||- Body
||||- VariableDeclaration (ar)
|||||- ArrayType
||||||- IntLiteral (3)
||||||- IntType
||||- VariableDeclaration (rec)
|||||- RecordType
||||||- VariableDeclaration (a)
|||||||- BoolType
|||||||- BoolLiteral (1)
||||||- VariableDeclaration (b)
|||||||- IntType
|||||||- IntLiteral (5)
||||||- VariableDeclaration (c)
|||||||- DoubleType
|||||||- DoubleLiteral (5.5)
```



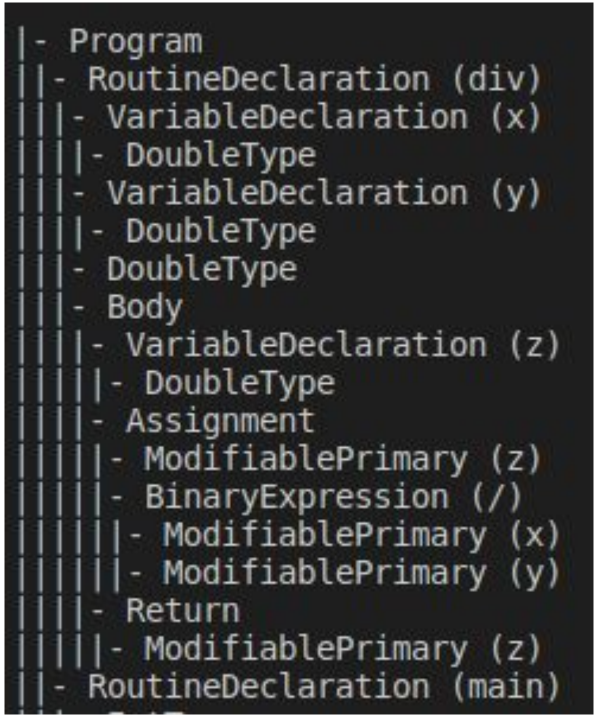
```
||||||- VariableDeclaration (c)
|||||||- DoubleType
|||||||- DoubleLiteral (5.5)
||||- ForLoop (i)
|||||- IntLiteral (1)
|||||- IntLiteral (3)
|||||- Body
||||||- Assignment
|||||||- ModifiablePrimary (ar)
||||||||- ModifiablePrimary (i)
|||||||- ModifiablePrimary (i)
||||- ForeachLoop (x)
|||||- ModifiablePrimary (rec)
|||||- Body
||||||- Print
|||||||- ModifiablePrimary (x)
||||- Return
|||||- IntLiteral (0)
```

Example #3 (Complex)

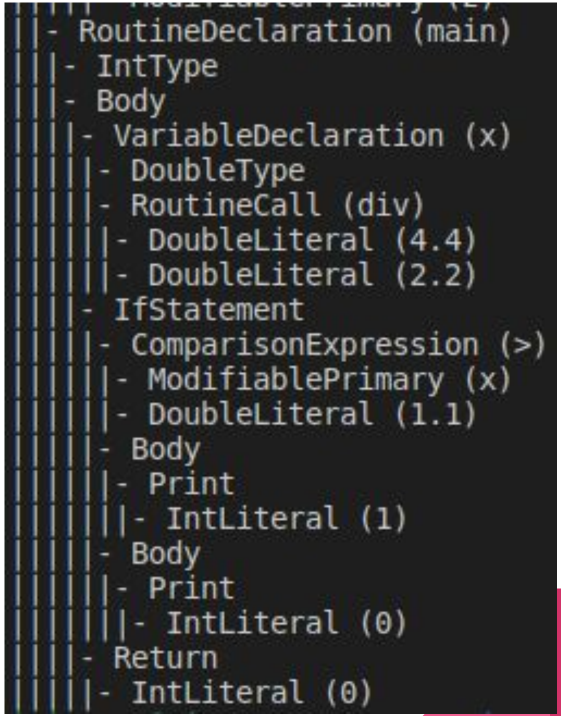


```
routine div (x : real, y : real) : real is

    var z : real;
    z := x / y;

    return z;
end

routine main () : integer is

    var x : real is div( 4.4 , 2.2 );

    if x > 1.1 then
        print(1);
    else
        print(0);
    end

    return 0;
end
```

Example #3 - AST Visualization



```
|- Program
||- RoutineDeclaration (div)
|||- VariableDeclaration (x)
||||- DoubleType
|||- VariableDeclaration (y)
||||- DoubleType
|||- DoubleType
|||- Body
||||- VariableDeclaration (z)
|||||- DoubleType
||||- Assignment
|||||- ModifiablePrimary (z)
|||||- BinaryExpression (/)
||||||- ModifiablePrimary (x)
||||||- ModifiablePrimary (y)
||||- Return
|||||- ModifiablePrimary (z)
||- RoutineDeclaration (main)
```



```
||- RoutineDeclaration (main)
|||- IntType
|||- Body
||||- VariableDeclaration (x)
|||||- DoubleType
|||||- RoutineCall (div)
||||||- DoubleLiteral (4.4)
||||||- DoubleLiteral (2.2)
||||- IfStatement
|||||- ComparisonExpression (>)
||||||- ModifiablePrimary (x)
||||||- DoubleLiteral (1.1)
|||||- Body
||||||- Print
|||||||- IntLiteral (1)
|||||- Body
||||||- Print
|||||||- IntLiteral (0)
||||- Return
|||||- IntLiteral (0)
```

Team responsibilities

Asem: Linking Lexer and Parser, AST.

Mosab: Parser, AST, Automation.

Menna: Parser, AST.

Jaffar: Parser, Visualization, Automation.

Let’s go for a tour!

Thank you!