Compilers – II: Code Gen

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

GOALS OF THE ASSIGNMENT

- Implement code generation for our language Tangent
- Choice of Intermediate Representation - LLVM IR

PROGRESS REPORT

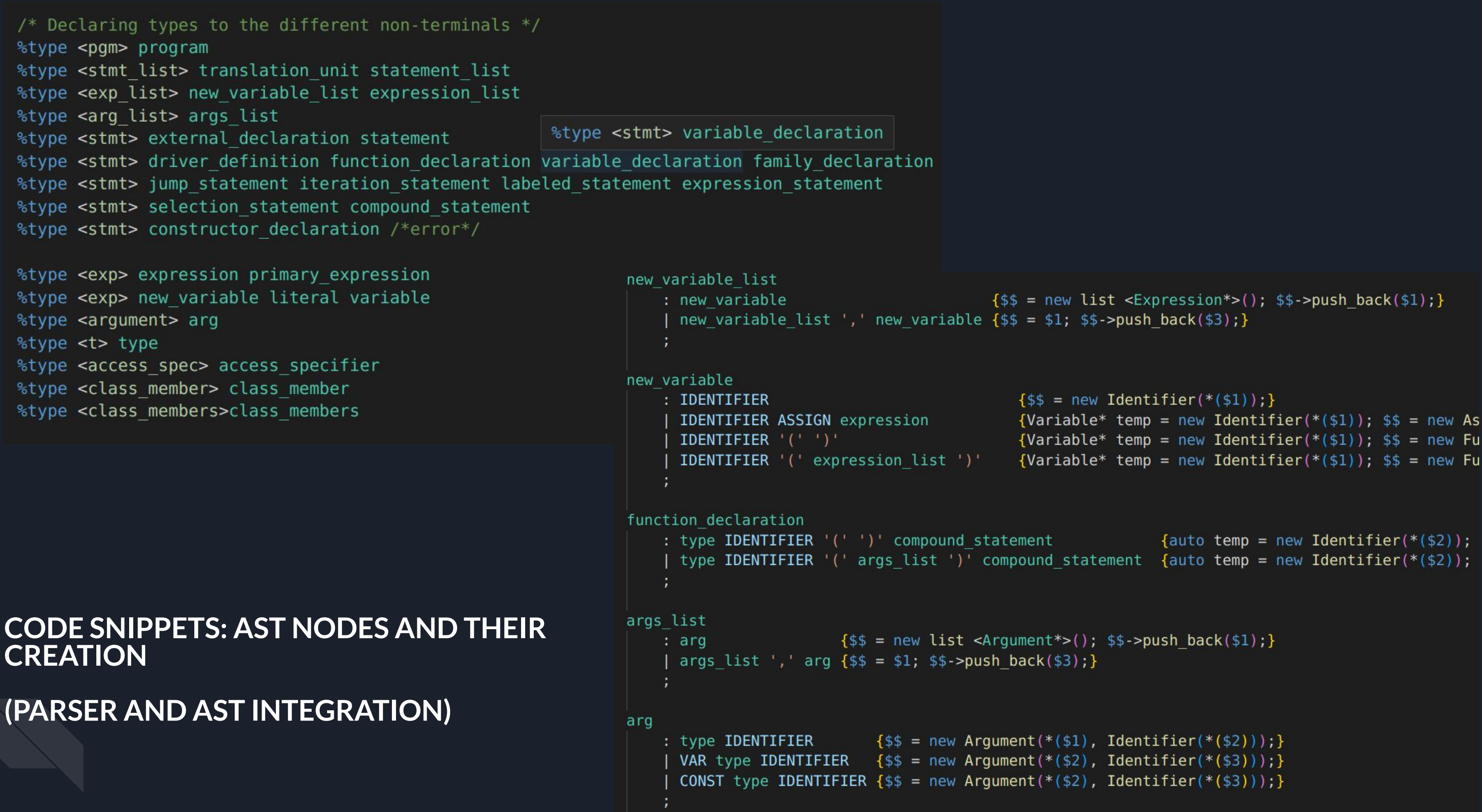
- We have successfully generated the Abstract Syntax Tree by creating AST nodes in the grammar rule actions.
- We have developed a basic understanding of how to generate LLVM IR from the AST.
- Implementation : by using LLVM libraries to generate specific LLVM IR for each of our AST nodes.
- We have started implementing this, but are yet to complete the code generation.

CHALLENGES FACED

- Scoping and user defined types in Symbol Table.
- Using C++ features with flex and bison.
- Merging multiple branches
- Passing the type names to the new variables during variable declaration.

CODE SNIPPETS: AST NODES AND THEIR CREATION

(PARSER AND AST INTEGRATION)

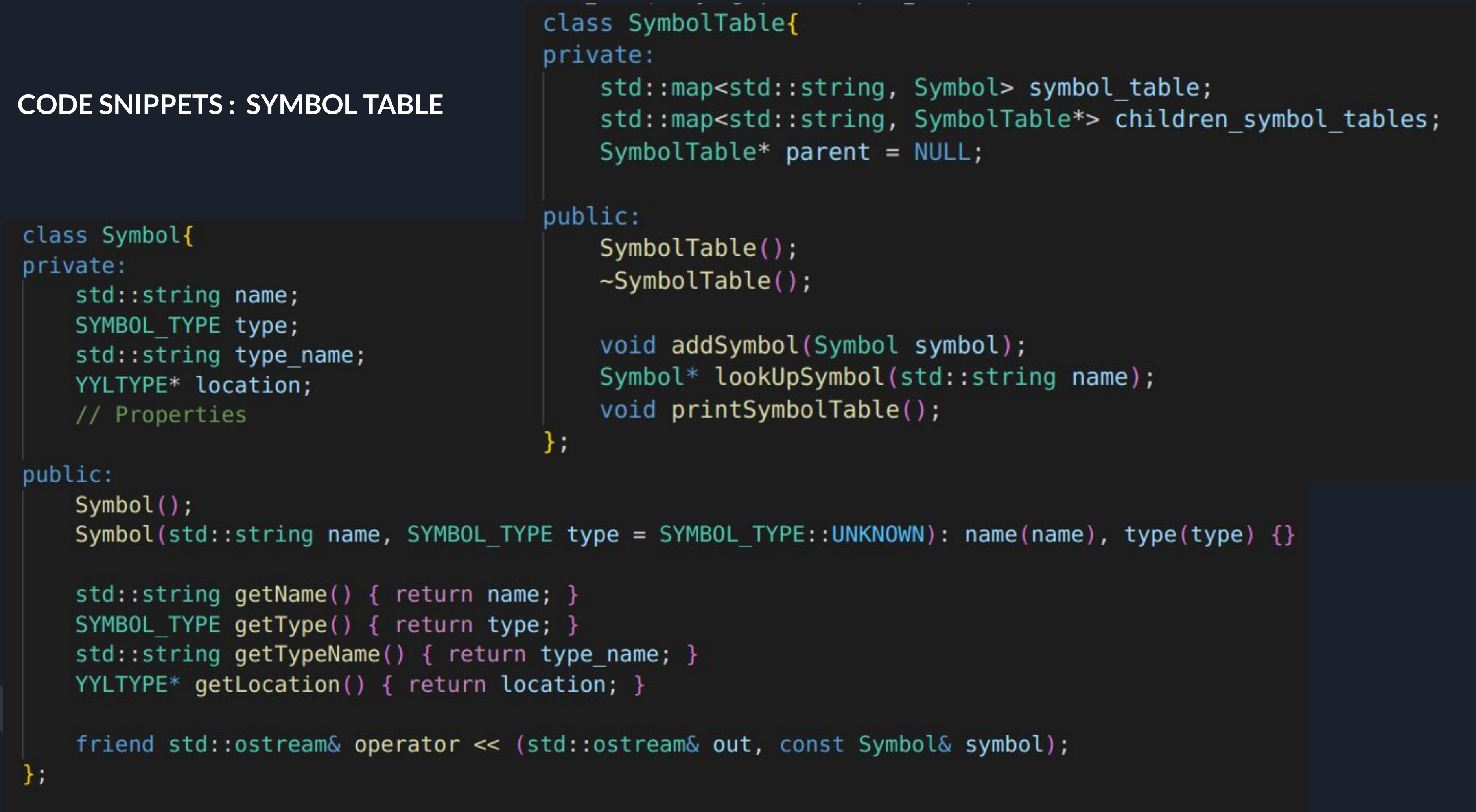


```
/* Declaring types to the different non-terminals */
%type <pgm> program
%type <stmt_list> translation_unit statement_list
%type <exp_list> new_variable_list expression_list
%type <arg_list> args_list
%type <stmt> external_declaration statement
%type <stmt> driver_definition function_declaration variable_declaration family_declaration
%type <stmt> jump_statement iteration_statement labeled_statement expression_statement
%type <stmt> selection_statement compound_statement
%type <stmt> constructor_declaration /*error*/

%type <exp> expression primary_expression
%type <exp> new_variable literal variable
%type <argument> arg
%type <t> type
%type <access_spec> access_specifier
%type <class_member> class_member
%type <class_members>class_members
```

```
%type <stmt> variable_declaration
```

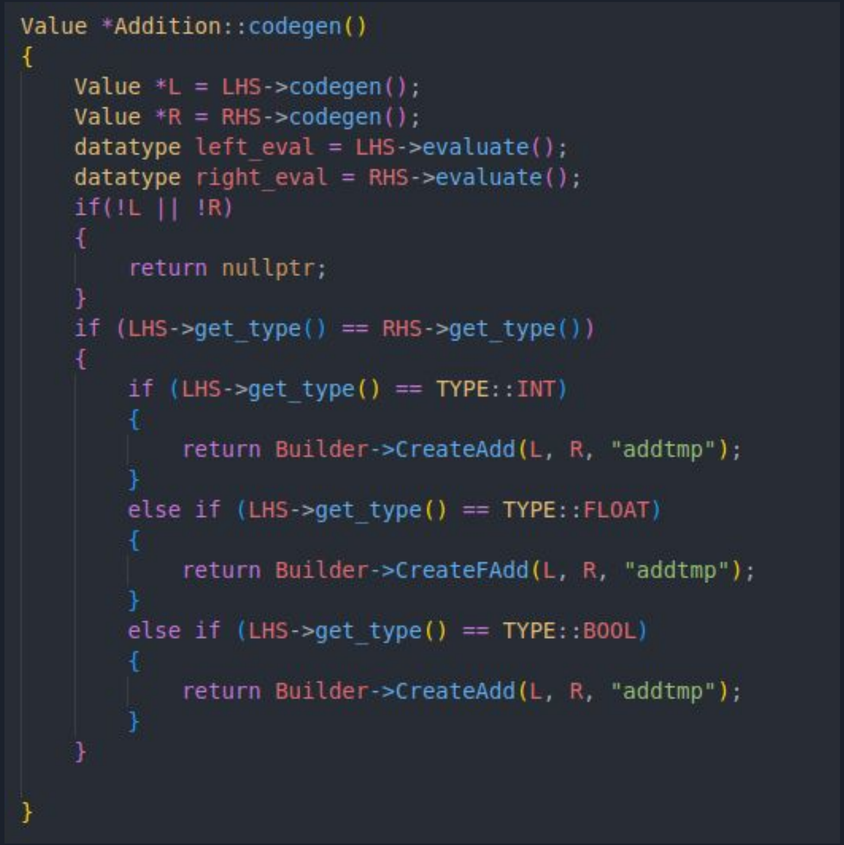
```
new_variable_list
    : new_variable                       {$$ = new list <Expression*>(); $$->push_back($1);}
    | new_variable_list ',' new_variable {$$ = $1; $$->push_back($3);}
    ;

new_variable
    : IDENTIFIER                            {$$ = new Identifier(*($1));}
    | IDENTIFIER ASSIGN expression          {Variable* temp = new Identifier(*($1)); $$ = new As
    | IDENTIFIER '(' ')'                    {Variable* temp = new Identifier(*($1)); $$ = new Fu
    | IDENTIFIER '(' expression_list ')'    {Variable* temp = new Identifier(*($1)); $$ = new Fu
    ;

function_declaration
    : type IDENTIFIER '(' ')' compound_statement            {auto temp = new Identifier(*($2));
    | type IDENTIFIER '(' args_list ')' compound_statement  {auto temp = new Identifier(*($2));
    ;

args_list
    : arg               {$$ = new list <Argument*>(); $$->push_back($1);}
    | args_list ',' arg {$$ = $1; $$->push_back($3);}
    ;

arg
    : type IDENTIFIER       {$$ = new Argument(*($1), Identifier(*($2)));}
    | VAR type IDENTIFIER   {$$ = new Argument(*($2), Identifier(*($3)));}
    | CONST type IDENTIFIER {$$ = new Argument(*($2), Identifier(*($3)));}
    ;
```

CODE SNIPPETS : SYMBOL TABLE



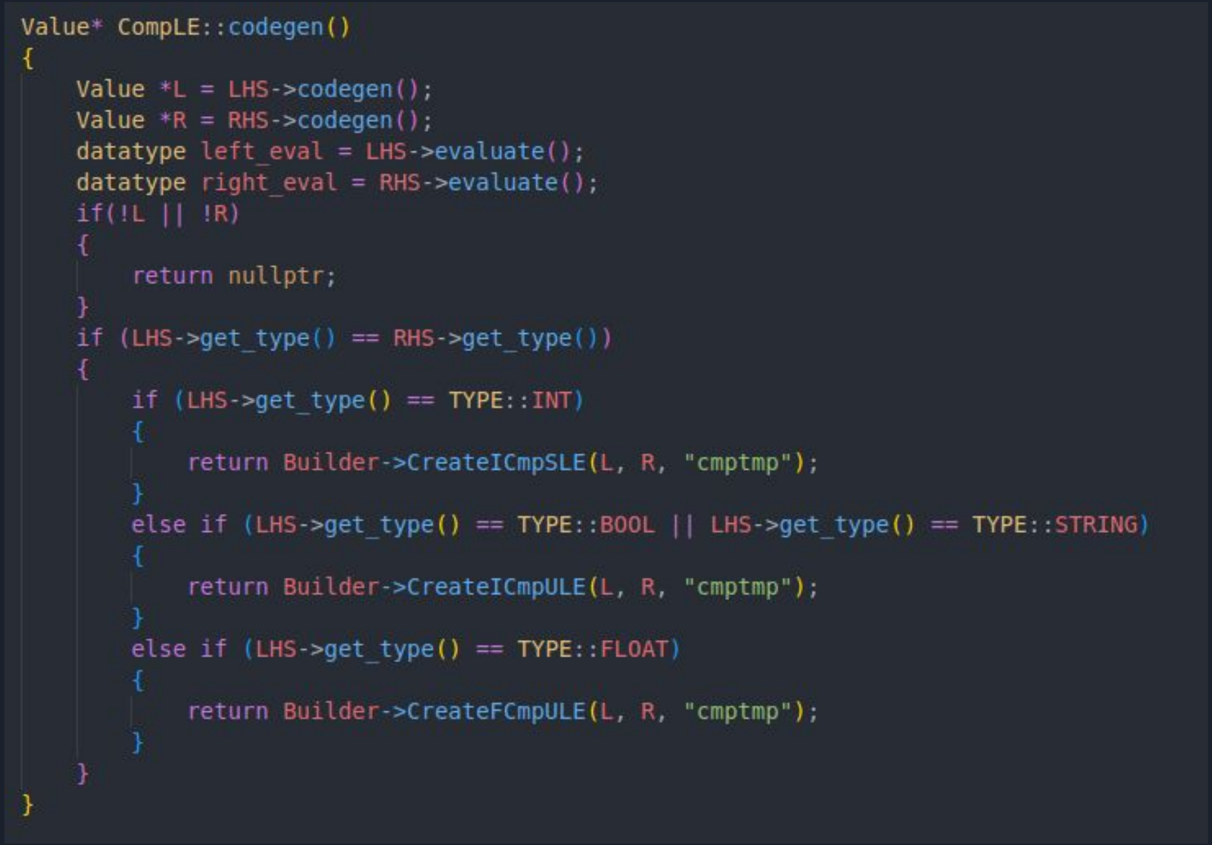
```
class SymbolTable{
private:
    std::map<std::string, Symbol> symbol_table;
    std::map<std::string, SymbolTable*> children_symbol_tables;
    SymbolTable* parent = NULL;

public:
    SymbolTable();
    ~SymbolTable();

    void addSymbol(Symbol symbol);
    Symbol* lookUpSymbol(std::string name);
    void printSymbolTable();
};
```

```
class Symbol{
private:
    std::string name;
    SYMBOL_TYPE type;
    std::string type_name;
    YYLTYPE* location;
    // Properties

public:
    Symbol();
    Symbol(std::string name, SYMBOL_TYPE type = SYMBOL_TYPE::UNKNOWN): name(name), type(type) {}

    std::string getName() { return name; }
    SYMBOL_TYPE getType() { return type; }
    std::string getTypeName() { return type_name; }
    YYLTYPE* getLocation() { return location; }

    friend std::ostream& operator << (std::ostream& out, const Symbol& symbol);
};
```



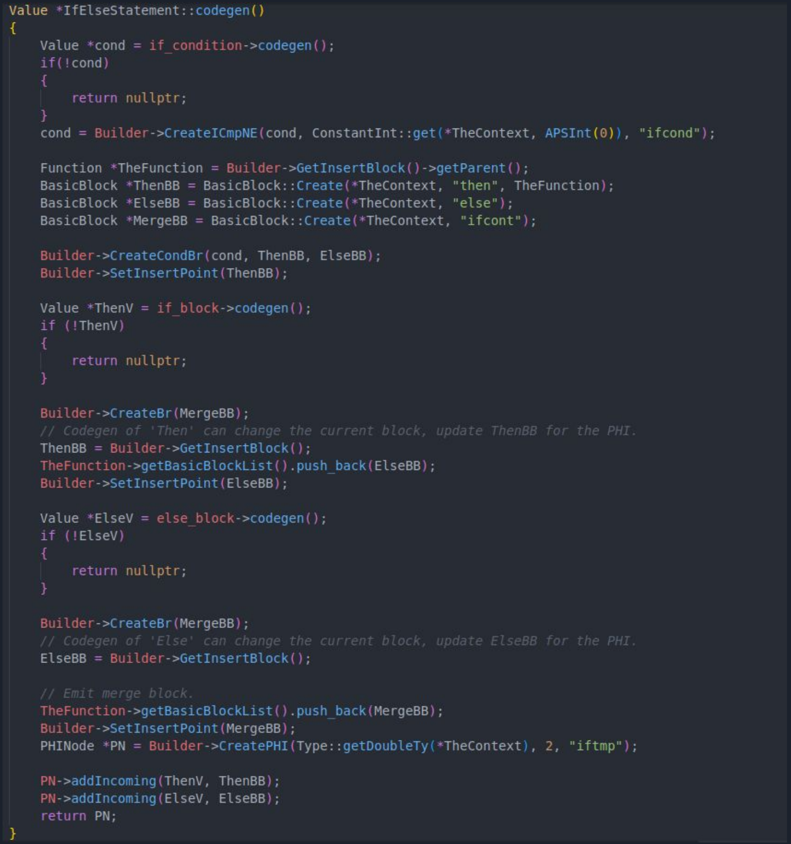
```
Value *Addition::codegen()
{
    Value *L = LHS->codegen();
    Value *R = RHS->codegen();
    datatype left_eval = LHS->evaluate();
    datatype right_eval = RHS->evaluate();
    if(!L || !R)
    {
        return nullptr;
    }
    if (LHS->get_type() == RHS->get_type())
    {
        if (LHS->get_type() == TYPE::INT)
        {
            return Builder->CreateAdd(L, R, "addtmp");
        }
        else if (LHS->get_type() == TYPE::FLOAT)
        {
            return Builder->CreateFAdd(L, R, "addtmp");
        }
        else if (LHS->get_type() == TYPE::BOOL)
        {
            return Builder->CreateAdd(L, R, "addtmp");
        }
    }

}
```

CODE SNIPPETS : CODEGEN FOR BINARY ARITHMETIC OPERATION

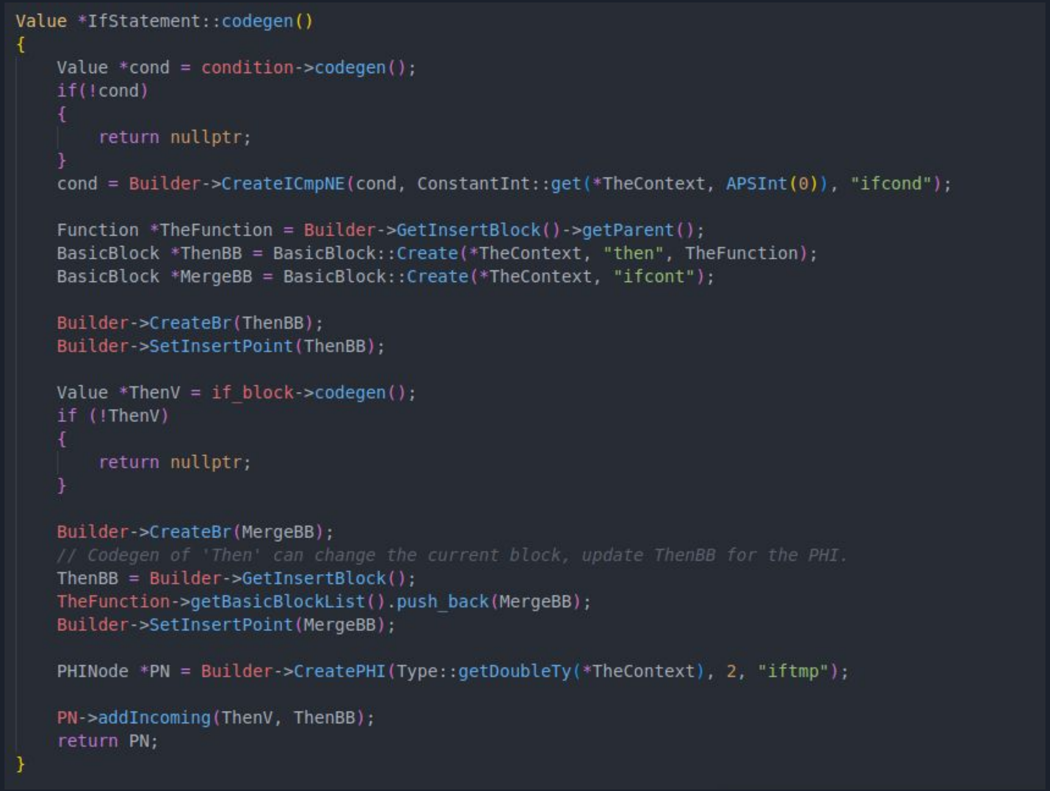


```
Value* CompLE::codegen()
{
    Value *L = LHS->codegen();
    Value *R = RHS->codegen();
    datatype left_eval = LHS->evaluate();
    datatype right_eval = RHS->evaluate();
    if(!L || !R)
    {
        return nullptr;
    }
    if (LHS->get_type() == RHS->get_type())
    {
        if (LHS->get_type() == TYPE::INT)
        {
            return Builder->CreateICmpSLE(L, R, "cmptmp");
        }
        else if (LHS->get_type() == TYPE::BOOL || LHS->get_type() == TYPE::STRING)
        {
            return Builder->CreateICmpULE(L, R, "cmptmp");
        }
        else if (LHS->get_type() == TYPE::FLOAT)
        {
            return Builder->CreateFCmpULE(L, R, "cmptmp");
        }
    }
}
```

CODE SNIPPETS : CODEGEN FOR BINARY LOGICAL OPERATION



```
Value *IfElseStatement::codegen()
{
    Value *cond = if_condition->codegen();
    if(!cond)
    {
        return nullptr;
    }
    cond = Builder->CreateICmpNE(cond, ConstantInt::get(*TheContext, APSInt(0)), "ifcond");

    Function *TheFunction = Builder->GetInsertBlock()->getParent();
    BasicBlock *ThenBB = BasicBlock::Create(*TheContext, "then", TheFunction);
    BasicBlock *ElseBB = BasicBlock::Create(*TheContext, "else");
    BasicBlock *MergeBB = BasicBlock::Create(*TheContext, "ifcont");

    Builder->CreateCondBr(cond, ThenBB, ElseBB);
    Builder->SetInsertPoint(ThenBB);

    Value *ThenV = if_block->codegen();
    if (!ThenV)
    {
        return nullptr;
    }

    Builder->CreateBr(MergeBB);
    // Codegen of 'Then' can change the current block, update ThenBB for the PHI.
    ThenBB = Builder->GetInsertBlock();
    TheFunction->getBasicBlockList().push_back(ElseBB);
    Builder->SetInsertPoint(ElseBB);

    Value *ElseV = else_block->codegen();
    if (!ElseV)
    {
        return nullptr;
    }

    Builder->CreateBr(MergeBB);
    // Codegen of 'Else' can change the current block, update ElseBB for the PHI.
    ElseBB = Builder->GetInsertBlock();

    // Emit merge block.
    TheFunction->getBasicBlockList().push_back(MergeBB);
    Builder->SetInsertPoint(MergeBB);
    PHINode *PN = Builder->CreatePHI(Type::getDoubleTy(*TheContext), 2, "iftmp");

    PN->addIncoming(ThenV, ThenBB);
    PN->addIncoming(ElseV, ElseBB);
    return PN;
}
```



```
Value *IfStatement::codegen()
{
    Value *cond = condition->codegen();
    if(!cond)
    {
        return nullptr;
    }
    cond = Builder->CreateICmpNE(cond, ConstantInt::get(*TheContext, APSInt(0)), "ifcond");

    Function *TheFunction = Builder->GetInsertBlock()->getParent();
    BasicBlock *ThenBB = BasicBlock::Create(*TheContext, "then", TheFunction);
    BasicBlock *MergeBB = BasicBlock::Create(*TheContext, "ifcont");

    Builder->CreateBr(ThenBB);
    Builder->SetInsertPoint(ThenBB);

    Value *ThenV = if_block->codegen();
    if (!ThenV)
    {
        return nullptr;
    }

    Builder->CreateBr(MergeBB);
    // Codegen of 'Then' can change the current block, update ThenBB for the PHI.
    ThenBB = Builder->GetInsertBlock();
    TheFunction->getBasicBlockList().push_back(MergeBB);
    Builder->SetInsertPoint(MergeBB);

    PHINode *PN = Builder->CreatePHI(Type::getDoubleTy(*TheContext), 2, "iftmp");

    PN->addIncoming(ThenV, ThenBB);
    return PN;
}
```

CODE SNIPPETS : CODEGEN FOR IF-ELSE STATEMENT