DataType

```
ScalarDataType type
vector<int> *dims
string type_str
```

ArrDecl →

ArrType →

ScalarType →

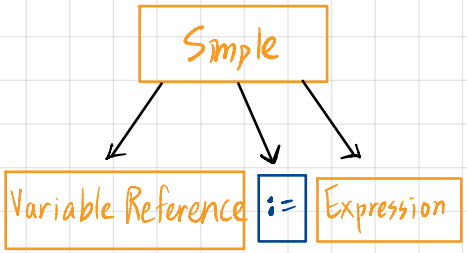
Type →

Declaration →

Assignment

```
VariableReferenceNode * variable
ExpressionNode * expr
```

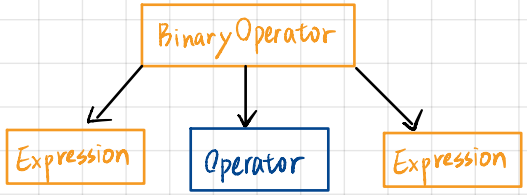
Simple → Variable Reference := Expression



BinaryOperator

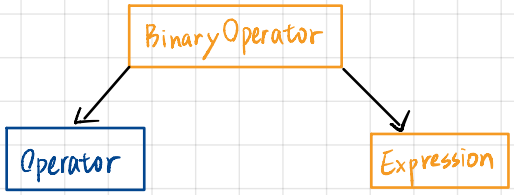
```
Operator * opr;
ExpressionNode *expr1;
ExpressionNode *expr2;
```

Expression → Expression
x ÷ % + − < <= > >= == <>
Expression
AND, OR



UnaryOperator

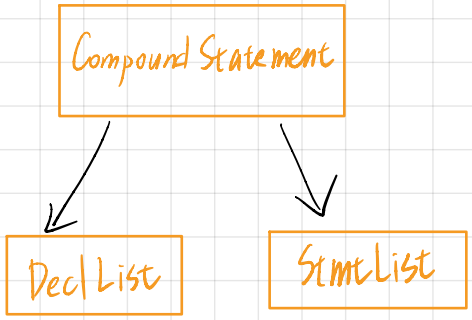
```
Operator * opr;
ExpressionNode *expr1;
```



Expression → NOT −
Expression

CompoundStatement

```
vector < DeclNode* > *declarations;
vector < AstNode* > *statements;
```



Compound Statement →

Constant Value

```
DataType type;
{ int    int_value
  float  real_value
  bool   bool_value
  string string_value
```

Literal Constant →

String and Boolean →

Integer and Real →

DeclNode

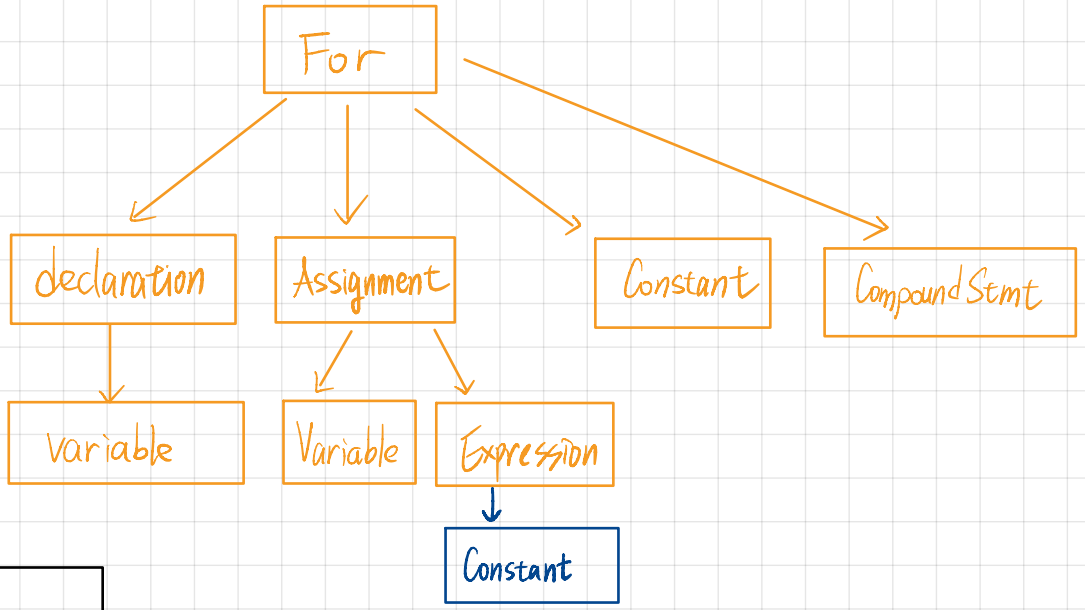
vector<VariableNode*> *variables
string type_str

Declaration →

Formal Arg →

For

DeclNode* decl
AssignmentNode* init
ConstantValueNode* condition
CompoundStatementNode* body



Function

string name
vector<DeclNode*> *declarations;
DataType *return_type;
CompoundStatementNode *compound;
string proto_str;

function declaration...

FunctionInvocation

string name
expression_list;

FunctionInv →

If

ExpressionNode* condition;
CompoundStmtNode* true_cmd;
CompoundStmtNode* false_cmd;

Location

uint32_t line
uint32_t col
~Location
Location(line, col)

AstNode

Location location
AstNode(line, col)
const Location& getLocation()
virtual void print()
virtual ~AstNode()
virtual void accept()
virtual void visitChildNodes()

① We need accept for others to visit.
② visitChildNodes to print its children.
③ visit() does these things:
a.) print the Node's data member.
b.) visit its child nodes.

AstDumper

void visit(...)

↳ overload for all kinds of nodes

ProgramNode

string name
vector<DeclNode*> *declarations
vector<FunctionNode*> *functions
CompoundStatementNode* body
~ProgramNode()
ProgramNode(...): AstNode() ...

AstNode Members

↳ omitted for simplicity

CompoundStatement

vector<DeclNode*> *declarations;
vector<AstNode*> *statements;

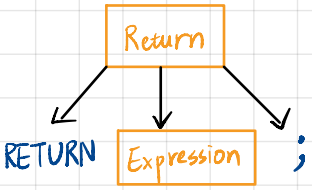
VariableNode

string Name
△ Type

ReturnNode

`ExpressionNode* expr;`

Return → RETURN Expression ;



VariableReference

`string name`

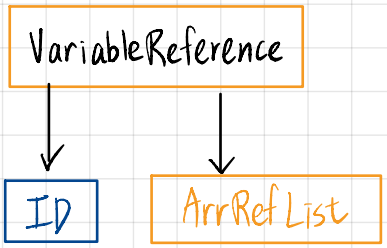
`vector<ExpressionNode*> *dimensions`

Variable Reference → ID Arr Ref List

Arr Ref List → ε | Arr Refs

Arr Refs → [Expression]

| Arr Refs [Expression]



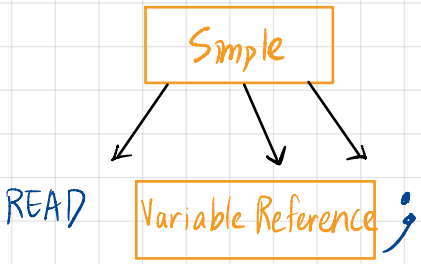
Read

VariableReferenceNode * variable

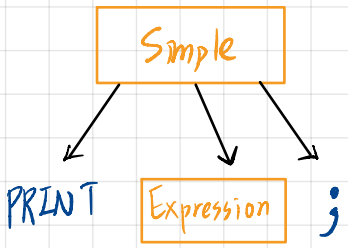
Print

ExpressionNode * variable

Simple → READ VariableReference ;



Simple → PRINT Expression ;



Function Invocation

string Name

vector<ExpressionNodes*>*Arguments

Function Call → Function Invocation ;

Function Invocation → ID (Expression List)

Expression List → ε | Expressions

Expressions → Expression | Expressions , Expression