Team Hmm

Semantic Analyzer

Imperative (I) Language

Team Members

Mikhail Trifonov

Kirill Efimovich

Technology Stack

Analyzer Implementation Details

Component	Technology
Source Language	Imperative (I)
Implementation Language	C++ (C++17)
Parser/AST	Bison/Flex + C++ AST
Integration Point	Runs after successful parse in `parser.y`
Output	Diagnostics + optimized AST (printed by existing printer)

Where the Analyzer Fits

Pipeline Overview

1. Parse (Bison/Flex) → build C++ AST
2. Semantic checks (non-mutating)
3. Optimizations (AST transforms)
4. Safety validation (post-optimization)
5. Print optimized AST / proceed to backend

Result type: `Analyzer::Result { errors, warnings, optimizationsApplied }`

Non-mutating Checks (1/2)

- Control flow conditions must be boolean
  - `if (cond)` and `while (cond)`
- Assignments
  - Type of right-hand side must be compatible with target
- Routine calls
  - Existence, arity, and per-parameter type compatibility

Non-mutating Checks (2/2)

- Records
  - Duplicate fields rejected; field lookup verified
- Arrays
  - Index must be `integer`
  - Static bounds warning when both size and index are constants
- Routine returns (arrow bodies)
  - `=> expr` type must match declared return type

Optimizations

- Constant folding
  - Arithmetic, relational, boolean, unary
- If simplification
  - Constant condition selects a branch
  - Declarations from chosen branch are hoisted and folded
- While-false elimination
  - Entire loop removed when condition is constant `false`
- Remove unused declarations (no initializer)

Constant Folding Examples

```
var a: integer is 5 + 3;      --→ a = 8
var b: boolean is not (true and false) or (1 < 2); --→ b = true
```

✓ Arithmetic, boolean, relational, and unary folds

If Simplification + Hoisting

Input:

```
if true then
  var x: integer is 1 + 1;
  print x;
end
```

After analyze:

- Branch flattened into the parent body
- `x` hoisted; initializer folded to `2`
- `print x` remains inlined

Nested Hoisting

Input:

```
if true then
  var x: integer is 1 + 1;
  if true then
    var y: integer is 2 + 2;
    print y;
  end
end
print x
```

Behavior:

- Inner constant-if pre-simplified before hoist
- Hoisted: `x = 2`, `y = 4`
- Statements spliced; `print y` preserved; `print x` valid at top level

Hoisting: Conflict Detection

Input:

```
var a: integer is 5 + 3;
if true then
  var a: integer is 1 + 2;
  print a;
end
```

Result:

- Error: `Duplicate variable declaration 'a' in same scope`
- No auto-renaming is performed

While False Elimination

Input:

```
while false loop
  var q: integer is 1 + 1;
  print q;
end
```

Behavior:

- Loop removed entirely
- No declarations hoisted from a dead loop body

Remove Unused Declarations

- Drops variables that are never referenced and have no initializer
- Preserves variables with initializers (possible side effects / clarity)
- Counts toward `optimizationsApplied`

Post-optimization Safety Check

- Re-scan top-level statements against program-level declarations
- Detects undefined identifiers introduced by control simplifications
- Example:

```
if false then var y: integer is 1; end
print y           --→ error: Undefined variable 'y'
```

Questions?