Definition

- Declaration table - to store variable declarations
- Files - to write the code
- Attribute types
    - `code` - File of code generated.
    - `error` - File of semantic errors.
    - `next` - Label of the next instruction on the code file.
    - `top` - Current (predicted) size of run-time stack.
    - `type` - Type of subtree. Used for type-checking.
- Functions
    - `enter(name, n)` - Binds "name" with stack location n. Returns n.
    - `lookup(name)` - Returns the location of "name". Returns 0 if "name" is not found.
    - gen(file, arg_1, ..., arg_n) - Writes a new line to "file". The line contains arg_1 , ..., arg_n. Returns the new, modified file.
    - `open()` - Creates a new file.
    - `close()` - Closes a file.
- Attributes
    - `S(program) = {code↑, error↑}`
    - `I(program) = {}`
    - `S(assign)  = {code↑, error↑, next↑, top↑, type↑}`
    - `I(assign)  = {code↓, error↓, next↓, top↓}`
- Convention

    All attributes except `type` attributes are both synthesized and inherited.
    a↑ is the synthesized attribute a.
    a↓ is the inherited attribute a.
    All other nodes except `program` node have the same synthesized and inherited attributes as `assign`.
    If axiom is missing assume

    - If no kids → a↑ (ε) = a↓ (ε)
    - Else → a↓ (1) = a↓ (ε), a↓ (i) = a↓ (i-1) for i < i ≤ n, a↑ (ε) = a↑ (n)

**Note that not all the AST grammar rules are considered.

Axioms

P → <'program' '<identifier:x>' E E E E E '<identifier:y>'>

```
code↓(2) = open
next↓(2) = 1
top↓(2) = 0

code↑(ε) = close(gen(code↑(6), "stop"))
```

E → <'consts' E+>

```
Use defaults
```

E → 'consts'

```
Use defaults
```

E → <'const' '<identifier>' E>

```
code↓(2) = code↓(ε)
next↓(2) = next↓(ε)
top↓(2) = top↓(ε)

code↑(ε) = code↑(2)
next↑(ε) = next↑(2)
top↑(ε) = top↑(2)
```

E → '<integer:n>'

```
code↑(ε) = gen(code↓(ε), "lit", "n")
next↑(ε) = next↓(ε) + 1
top↑(ε) = top↓(ε) + 1
```

E → '<char:c>'

```
code↑(ε) = gen(code↓(ε), "lit", "c")
next↑(ε) = next↓(ε) + 1
top↑(ε) = top↓(ε) + 1
```

E → <'types' E+>

Use defaults

E → 'types'

Use defaults

E → <'type' '<identifier>' E>

```
code↓(2) = code↓(ε)
next↓(2) = next↓(ε)
top↓(2) = top↓(ε)

code↑(ε) = code↑(2)
next↑(ε) = next↑(2)
top↑(ε) = top↑(2)
```

E → <'dclns' E+>

Use defaults

E → 'dclns'

Use defaults

E → <'var' '<identifier>'+ '<identifier>'>

```
code↑(ε) = code↓(ε)
next↑(ε) = next↓(ε)
top↑(ε) = top↓(ε)
```

E → <'block' E+>

Use defaults

E → <'output' E+> (here 2 <= i <= n and assume only integer output)

```
code↓(1) = code↓(ε)
next↓(1) = next↓(ε)
top↓(1) = top↓(ε)

code↓(i) = gen(code↑(i-1), "print")
next↓(i) = next↑(i-1) + 1
top↓(i) = top↑(i-1) - 1

code↑(ε) = gen(code↑(n), "print")
next↑(ε) = next↑(n) + 1
```

```
top↑(ε) = top↑(n) - 1
```

E → <'if' E E E?>

E → <'if' E E>

```
code↓(2) = gen(code↑(1), "iffalse", next↑(2))
next↓(2) = next↑(1) + 1
top↓(2) = top↑(1) - 1
```

E → <'if' E E E>

```
code↓(2) = gen(code↑(1), "iffalse", next↑(2) + 1)
next↓(2) = next↑(1) + 1
top↓(2) = top↑(1) - 1

code↓(3) = gen(code↑(2), "goto", next↑(3))
next↓(3) = next↑(2) + 1
```

E → <'while' E E>

```
code↓(2) = gen(code↑(1), "iffalse", next↑(2) + 1)
next↓(2) = next↑(1) + 1
top↓(2) = top↑(1) - 1

code↑(ε) = gen(code↑(2), "goto", next↓(ε))
next↑(ε) = next↑(2) + 1
```

E → <'repeat' E+ E> (here 2 <= i <= n)

```
code↓(1) = code↓(ε)
next↓(1) = next↓(ε)
top↓(1) = top↓(ε)

code↓(i) = code↑(i-1)
next↓(i) = next↑(i-1)
top↓(i) = top↑(i-1)

code↑(ε) = gen(code↑(n), "iffalse", next↓(ε))
next↑(ε) = next↑(n) + 1
top↑(ε) = top↑(n) - 1
```

E → <'read' '<identifier:x>'+> (assume only integer input)

```
tempCode = code↑(ε)
```

```
tempTop = top↑(ε)

for each <identifier:x>:
     if lookup("x") = 0:
          enter("x", ++tempTop)
          tempCode = gen(tempCode, "read")
     else:
          tempCode = gen(gen(tempCode, "read"), "save", lookup("x"))

code↑(ε) = tempCode
next↑(ε) = next↓(ε) + 2 * n
top↑(ε) = top↓(ε) + # un-assigned identifiers
```

E → '<null>'

Use defaults

E → <'integer' E>

Use defaults

E → <'string' E>

Use defaults

E → '<string:s>'

```
code↑(ε) = gen(code↓(ε), "lit", "s")
next↑(ε) = next↓(ε) + 1
top↑(ε) = top↓(ε) + 1
type↑(ε) = "string"
```

E → <'assign' '<identifier:x>' E>

```
code↑(ε) = if lookup("x") = 0 then enter("x", top↑(2)); code↑(2)
else gen(code↑(2), "save", lookup("x"))
next↑(ε) = if lookup("x") = 0 then next↑(2) else next↑(2) + 1
top↑(ε) = if lookup ("x") = 0 then top↑(2) else top↑(2) - 1
```

E → <'swap' '<identifier:x>' '<identifier:y>'>

```
code↑(ε) = gen(gen(gen(gen(code↓(ε), "load", lookup("x")), "load",
lookup("y")), "save", lookup("x")), "save", lookup("y"))
next↑(ε) = next↓(ε) + 4
```

E → <'<=' E E> (assume only integer comparison)

```
code↑(ε) = gen(gen(code↑(2), "greaterthan"), "not")
next↑(ε) = next↑(2) + 2
top↑(ε) = top↑(2) - 1
```

E → <'<' E E> (assume only integer comparison)

```
code↑(ε) = gen(code↑(2), "lessthan")
next↑(ε) = next↑(2) + 1
top↑(ε) = top↑(2) - 1
```

E → <'>=' E E> (assume only integer comparison)

```
code↑(ε) = gen(gen(code↑(2), "lessthan"), "not")
next↑(ε) = next↑(2) + 2
top↑(ε) = top↑(2) - 1
```

E → <'>' E E> (assume only integer comparison)

```
code↑(ε) = gen(code↑(2), "greaterthan")
next↑(ε) = next↑(2) + 1
top↑(ε) = top↑(2) - 1
```

E → <'=' E E>

```
code↑(ε) = gen(code↑(2), "equal")
next↑(ε) = next↑(2) + 1
top↑(ε) = top↑(2) - 1
```

E → <'<>' E E>

```
code↑(ε) = gen(gen(code↑(2), "equal"), "not")
next↑(ε) = next↑(2) + 2
top↑(ε) = top↑(2) - 1
```

E → <'+' E E> (assume only integer addition)

```
code↑(ε) = gen(code↑(2), "add")
next↑(ε) = next↑(2) + 1
top↑(ε) = top↑(2) - 1
```

E → <'-' E E> (assume only integer subtraction)

```
code↑(ε) = gen(code↑(2), "subtract")
next↑(ε) = next↑(2) + 1
top↑(ε) = top↑(2) - 1
```

E → <'or' E E>

```
code↑(ε) = gen(code↑(2), "or")
next↑(ε) = next↑(2) + 1
top↑(ε) = top↑(2) - 1
```

E → <'*' E E> (assume only integer multiplication)

```
code↑(ε) = gen(code↑(2), "multiply")
next↑(ε) = next↑(2) + 1
top↑(ε) = top↑(2) - 1
```

E → <'/' E E> (assume only integer division)

```
code↑(ε) = gen(code↑(2), "divide")
next↑(ε) = next↑(2) + 1
top↑(ε) = top↑(2) - 1
```

E → <'and' E E>

```
code↑(ε) = gen(code↑(2), "and")
next↑(ε) = next↑(2) + 1
top↑(ε) = top↑(2) - 1
```

E → <'mod' E E> (assume only integer modulo operation)

```
code↑(ε) = gen(code↑(2), "mod")
next↑(ε) = next↑(2) + 1
top↑(ε) = top↑(2) - 1
```

E → <'-' E> (assume only integer comparison)

```
code↑(ε) = gen(code↑(1), "negate")
next↑(ε) = next↑(1) + 1
```

E → <'not' E>

```
code↑(ε) = gen(code↑(1), "not")
next↑(ε) = next↑(1) + 1
```

E → '<identifier:x>'

```
code↑(ε) = gen(code↓(ε), "load", lookup("x"))
next↑(ε) = next↓(ε) + 1
top↑(ε) = top↓(ε) + 1
```