

PROJECT Specification:  
Task Booklet 02

---

COS341 2020

# STATIC ANALYSIS

Now you will be working on the **static semantics** of SPL. You will build on your previous work in which your parser constructed an AST. Static semantics analysis involves traversing the AST to figure out that certain properties defined for SPL hold true in the parsed input program.

There will be 3 sub-parts of static semantics analysis: In this sub-part you will focus on the **scope of variables and procedures** (scoping). In later sub-parts you will look at type checking and value-definedness.

---

## SCOPING RULES

The scopes for SPL are **statically** defined as follows:

- ▶ Top level code is in scope **0**.
- ▶ Each procedure declaration opens a new scope **n+1** where **n** is the scope level the procedure is declared in.
- ▶ A variable or procedure name may be “*seen*” in the direction “*from inside out*” if it is referred to within a scope with the same or higher level scope than its declaration level – unless the same name is newly defined within this scope itself.

---

# SCOPING RULES: EXAMPLE


```
num app;  
app = 100;  
num car;  
car = 300;
```

```
proc alpha {  
  app = 110;  
  num bat;  
  bat = 210;
```

```
  proc kappa {  
    num app;  
    app = 111  
  }
```

```
  proc epsilon {  
    bat = 212;  
    car = 312  
  }  
}
```

```
proc beta {  
  app = 120;  
  bat = 220  
}
```

 bat is not declared

**Note:** Variables that refer to the same memory location have the same colour as each other.

---

## YOUR TASK:

- ▶ Write a program that **traverses the AST and determines the scope to which each node belongs.**
  - ▶ Each AST node should be assigned a scope level.
  - ▶ At each **usage** of a variable or procedure **name**, **determine its corresponding declaration node:**
    - ▶ Is the corresponding declaration found in the same scope?
    - ▶ Or is the declaration “seen” from a (higher) outer scope?
- ▶ All this information should be stored in your **symbol table**, the fields of which are linked to the **AST nodes by means of “pointers”**.

---

## YOUR TASK

- ▶ When the scope information has been figured out, you must rewrite variable and procedure names uniquely as follows:
  - ▶ All instances of the same variable/procedure name should be given the same unique name throughout the program which differs from all other names.
  - ▶ All variables should be named with the scheme **V0, V1, ...**
    - ▶ Attention: When there were **two “x” in different scopes**, they are actually **two different variables** and will thus obtain different new names!
  - ▶ All procedures should be named with the scheme **P0, P1, ...**
  - ▶ Undefined variables or procedures should be named **U**

---

## EXAMPLE (continued... After re-naming)

```
num V0;  
V0 = 100;  
num V1;  
V1 = 300;
```

```
proc P0 {  
  V0 = 110;  
  num V2;  
  V2 = 210;
```

```
  proc P2 {  
    num V3;  
    V3 = 111  
  }
```

```
  proc P3 {  
    V2 = 212;  
    V1 = 312  
  }  
}
```

```
proc P1 {  
  V0 = 120;  
  U = 220  
}
```

Important:

---

## ADDITIONAL NOTES

- ▶ **Plagiarism is not allowed!** You may not use any code you have not written yourself.
- ▶ Both your AST and symbol table will be checked in the demo.
  - ▶ Your symbol table must have both scope-IDs and edited variable and procedure names.
  - ▶ In the demo you must be able to show the links between AST nodes and symbol table information.

---

And now: HAPPY CODING 😊 😊 😊