Tutorial #6 - Names, Bindings and Scopes

Habib Ghaffari

<2022-10-31 Mon>

Contents

1 Introduction

Today, we are going to review the topics presented in chapter 5. Then we will try to solve some sample examples that will help you answer some of the questions presented in assignment #2.

2 Names

A name is a string of characters used to identify some entity in a program. Names in most programming languages have the same form: a letter followed by a string consisting of letters, digits, and underscore characters `( _ )`.

Examples:

`Haskell`

```
<H_ID> ::= <head> {<tail>}
<head> ::= <upper> | <lower>
<tail> ::= <upper> | <lower> | <special> | <digits> | ’
<lower> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s
<upper> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S
<digits> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Two main primary issues with naming are?

- Case sensitivity
- Special words
- Question:
  - What is a reserved word?
- Exercise: Look at the following names:
  1. FirstName
  2. $\text{First}_{\text{Name}}$
  3. firstname
  4. firstName
  5. Which one is better in terms of Readability?
  6. Which one is better in terms of Writability?
  7. Which one is accepted in C programming language?
  8. Which one is better in Python programming language?
  9. Which one is *cammel notation*?
  10. Which one is not popular anymore?

A program variable is an abstraction of a computer memory cell or collection of cells.

Variables can be characterized as a sextuple of attributes:

- Name
- Address: The memory address with which it is associated
- Value: The contents of the location with which the variable is associated

- Type: determines the range of values of variables and the set of operations that are defined for values of that type; in the case of floating point, type also determines the precision
- Lifetime [Discuss later]
- Scope [Discuss later]

3 Binding

A binding is an association between an entity and an attribute, such as between a variable and its type or value, or between an operation and a symbol. Binding time is the time at which a binding takes place.

- A binding is static if it first occurs before run time and remains unchanged throughout program execution.
  - Type may be specified by explicit or implicit deceleration:
    - Question: Bring one example for each of the above decelerations:

```
var sum = 0;
var total = 0.0;
var name = "Fred";
```

- A binding is dynamic if it first occurs during execution or can change during execution of the program

3.1 Storage Bindings and Lifetime

4 Scope and Lifetime

The scope of a variable is the range of statements over which it is visible.

The local variables of a program unit are those that are declared in that unit. The nonlocal variables of a program unit are those that are visible in the unit but not declared there. Global variables are a special category of nonlocal variables

4.1 Static Scope

The general rule for tracing visibility of a variable:

- Search declarations, first locally, then in increasingly larger enclosing scopes, until one is found for the given name
- Enclosing static scopes (to a specific scope) are called its static ancestors; the nearest static ancestor is called a static parent
- Some languages allow nested subprogram definitions, which create nested static scopes
- Variables can be hidden from a unit by having a "closer" variable with the same name

```
function big() { function sub1() {
var x = 7;
sub2(); }
function sub2() { var y = x;
}
var x = 3; sub1();
}
```

What if we consider the static scope rules to trace the visibility of `x`:

4.2 Blocks

- The scopes created by blocks, which could be nested in larger blocks, are treated exactly like those created by subprograms.

```
void sub() {
   int count=1;
    while (count <=100) {
      int count=2;
      count++;
      ...
   }
   printf(count);
   ...
}
```

- In `Let` Scheme:

```
In Scheme:
(LET (
   (name1 expression1) ...
   (namen expressionn)
)
```

4.3 Dynamic Scope

General rule for tracing visibility in dynamic scope:

- Based on calling sequences of program units, not their textual layout
- References to variables are connected to declarations by searching back through the chain of subprogram calls that forced execution to this point

```
function big() {
    function sub1() {
      var x = 7;
      sub2();
    }
    function sub2() { var y = x;
    }
    var x = 3;
    sub1();
}
```

Consider tracing this code based on dynamic scope rules. What would be the reference point for `x`?

When it comes to lifetime, it becomes a bit confusing. Even though it seems that these two are related to each other but they are totally different concepts.

4.4 Referencing Enviroments

The referencing environment of a statement is the collection of all variables that are visible in the statement. The referencing environment of a statement in a static- scoped language is the variables declared in its local scope plus the collection of all variables of its ancestor scopes that are visible.

```
g = 3; # A global
def sub1():
a = 5; # Creates a local
b = 7; # Creates another local
. . . <------------------------------ 1
def sub2():
global g; # Global g is now assignable here
c = 9; # Creates a new local
. . . <------------------------------ 2 def sub3():
nonlocal c: # Makes nonlocal c visible here g = 11; # Creates a new local
. . . <------------------------------ 3
```

The referencing environments of the indicated program points are as follows:

pints	Variables	Where Declared
1)	a b	sub 1
	g	g for referencing
2)	c	sub2
	g	
3)	c	nonlocal sub2
	g	sub 3 (local)

Let's have a look at another example:

```
void sub1() {
  int a, b;
  . . . <------------ 1
 } /* end of sub1 */
void sub2() {
  int b, c;
 . . . . <------------ 2 sub1();
 } /* end of sub2 */
void main() {
 int c, d;
 . . . <------------ 3
 sub2();
  }  /* end of main */
```

Consider the following example program. Assume that the only function calls are the following: `main` calls `sub2`, which calls `sub1`.

The referencing environments of the indicated program points are as follows:

pints	Variables	Where Declared
1)	a b	sub1
	c	sub2
	d	main
	b c of sub2	hidden
	b of main	hidden
2)	b c	sub2
	d	main
	c of main	hidden
3)	c d	main

5 Questions:

1. Consider the following JavaScript skeletal program:

```
// The main program
var x;
function sub1(){
    var x:
    function sub2(){
        . . .
    }
}
function sub3(){
. . .
}
```

Assume that the execution of this program is in the following unit order:

- `main` calls `sub1`
- `sub1` calls `sub2`
- `sub2` calls `sub3`

Assuming static scoping, in the following, which declaration of `x` is the correct one for a reference to `x`?

i. `sub1`, Why? (sub1) ii. `sub2`, Why? (sub1) iii. `sub3`, Why? (main)
Repeat part a, but assume dynamic scoping:
i. `sub1`, Why? (sub1) ii. `sub2`, Why? (sub1) iii. `sub3`, Why? (sub1)

1. Let's write a peace of code in `JavaScript`:

```
var x;
function sub1() {
 document.write("x = " + x + "");
}
function sub2() {
 var x;

 x = 10;
 sub1();
}
x = 5;
sub2();
```

- What value of `x` is displayed if we use static-scope rules? Why?

  x = 5

- What if we use dynamic-scope rules? Why?

  x = 10

- Let's write another piece of code in `JavaScript`:

```
var x, y, z;
function sub1() {
var a, y, z;
  function sub2() {
   var a, b, z;
   . . .
  }
  . . .
}
function sub3() {
  var a, x, w;
  . . .
}
```

List all the variables, along with the program units where they are declared, that are visible in the bodies of `sub1`, `sub2`, and `sub3`, assuming static scoping is used. Justify your answer:

- In sub1: a sub1 y sub1 z sub1 x main
- In sub2: a sub2 b sub2 z sub2 y sub1 x main
- In sub3: a sub3 x sub3 w sub3 y main z main
- Consider the following `C` program:

```
void fun(void) {
int a, b, c; /* definition 1 */
. . .
while (. . .) {
int b, c, d; /*definition 2 */
. . . <------------- 1
while (. . .) {
int c, d, e; /* definition 3 */
. . . <------------- 2
}
. . . <-------------- 3
}
. . . <---------------- 4
}
```

For each of the four marked points in this function, list each visible variable, along with the number of the definition statement that defines it. Justify your answer?

- Point 1: a 1 b 2 c 2 d 2
- Point 2: a 1 b 2 c 3 d 3 e 3
- Point 3: a 1 b 2 c 2 d 2

Point 4: a 1 b 1 c 1

1. Consider the following skeletal `C` program:

```
void fun1(void); /* prototype */
void fun2(void); /* prototype */
void fun3(void); /* prototype */
void main() {
int a, b, c;
. . .
}
void fun1(void) {
int b, c, d;
. . .
}
void fun2(void) {
int c, d, e;
. . .
}
void fun3(void) {
int d, e, f;
. . .
}
```

Given the following calling sequences and assuming that dynamic scoping is used, what variables are visible during execution of the last function called? Include with each visible variable the name of the function in which it was defined. Justify your answer?

a. `main` calls `fun1`; `fun1` calls `fun2`; `fun2` calls `fun3`. b. `main` calls `fun1`; `fun1` calls `fun3`. c. `main` calls `fun2`; `fun2` calls `fun3`; `fun3` calls `fun1`. d. `main` calls `fun3`; `fun3` calls `fun1`. e. `main` calls `fun1`; `fun1` calls `fun3`; `fun3` calls `fun2`. f. `main` calls `fun3`; `fun3` calls `fun2`; `fun2` calls `fun1`.

Question	Variables	Where Declared
a)	d e f	fun3
	c	fun2
	b	fun1
	a	main
b)	d e f	fun3
	b c	fun1
	a	main
c)	b c d	fun1
	e f	fun3
	a	main
d)	b c d	fun1
	e f	fun3
	a	main
e)	c d e	fun2
	f	fun3
	b	fun1
	a	main
f)	b c d	fun1
	e	fun2
	f	fun3
	a	main

1. Consider the following program, written in `JavaScript`-like

syntax:

```
// main program
var x, y, z;
function sub1() {
var a, y, z;
. . .
}
function sub2() {
var a, b, z;
. . .
}
function sub3() {
var a, x, w;
. . .
}
```

Given the following calling sequences and assuming that dynamic scoping is used, what variables are visible during execution of the last subprogram activated? Include with each visible variable the name of the unit where it is declared.

a. `main` calls `sub1`; `sub1` calls `sub2`; `sub2` calls `sub3`. b. `main` calls `sub1`; `sub1` calls `sub3`. c. `main` calls `sub2`; `sub2` calls `sub3`; `sub3` calls `sub1`. d. `main` calls `sub3`; `sub3` calls `sub1`. e. `main` calls `sub1`; `sub1` calls `sub3`; `sub3` calls `sub2`. f. `main` calls `sub3`; `sub3` calls `sub2`; `sub2` calls `sub1`.

Question	Variables	Where Declared
a)	a x w	sub3
	b z	sub2
	y	sub1
b)	a x w	sub3
	y z	sub1
c)	a y z	sub1
	x w	sub3
	b	sub2
d)	a y z	sub1
	x w	sub3
e)	a b z	sub2
	x w	sub3
	y	sub1
f)	a y z	sub1
	b	sub2
	x w	sub3