Department of Computer Engineering
University of Peradeniya
CO523 - Programming Languages

Lab 04: Demonstrating Static vs Dynamic Scoping

Objectives

- Understand the concepts of names, bindings, and scope in programming languages.
- Clearly distinguish between static (lexical) scoping and dynamic scoping.
- Predict program behavior based on scoping rules.
- Implement and experiment with code examples that demonstrate scoping differences.
- Develop practical programming skills through hands-on coding tasks.
- Analyze and explain variable binding behavior in different execution contexts.

1. Background Concepts

1.1. Names

A **name** is an identifier used to refer to variables, functions, constants, or other entities in a program.

e.g.;

```
int count;
float average;
```

Here, `count` and `average` are names.

1.2. Bindings

A **binding** is an association between a name and an entity (such as a value, variable, or function).

e.g.;

```
x = 10
```

The name `x` is bound to the value `10`

2. Scope

A **scope** is the largest program region where no bindings are changed.

The scope of a variable x in the region of the program in which the use of x refers to its declaration. One of the basic reasons for scoping is to keep variables in different parts of the program distinct from one another.

Scoping is generally divided into two classes:.

1. Static Scoping
2. Dynamic Scoping

2.1. Static Scoping

Static scoping is also called **lexical scoping**. In this scoping, a variable always refers to its top-level environment. This is a property of the program text and is unrelated to the run-time call stack. The scope is determined at compile-time based on the physical structure of the source code.

Most modern languages such as C, C++, Java, and Python use static scoping.

e.g.; *static scoping in python*

```
x = 10

def print_x():
  print("Static x:",x)

def main():
  x = 20
  print_x()

main()
```

Run the above given python code.

Here, printX() refers to `x` defined globally. The local `x` in `main()` does not affect `printX()`. Binding is determined by code structure, not call order

Note:

***Perl** is an excellent choice for practising this concept because it is one of the few languages that natively supports both static (lexical) and dynamic scoping.*

In Perl, the `'my'` keyword creates a static variable, while the `'local'` keyword (despite its name) creates a dynamic variable.

You can use an online compiler to test your Perl code (for example: https://onecompiler.com/perl).

Below is the same code written in the Perl language.

```perl
$x = 10;

sub print_x {
    print "Static x: $x\n";
}

sub main {
    my $x = 20; # 'my' creates a new lexical variable x
    print_x();
}

main();
```

Even though `main` defines an `x = 20`, that variable is only visible inside the `main` block. When `print_x` executes, it refers back to the global `$x` defined at the top of the script because that is where `print_x` was originally declared.

2.2. Dynamic Scoping

In **dynamic scoping**, the scope of a variable is determined by the call stack at runtime.

Used in older languages, like Lisp, and initially in Perl. Perl now has a way of specifying static scope as well.

Refer to the following pseudo code.

```
x = 10

function print_X() {
print x
}

function main() {
x = 20
print_X()
}
```

What should be the output of this under Dynamic Scoping?

It should be 20.

Why?

Dynamic scoping determines the scope based on the calling sequence (the "runtime stack"). When `print_x` is called, it looks for `x` in the environment that called it.

Perl Implementation.

```
$x = 10;

sub print_x {
    print "Dynamic x: $x\n";
}

sub main {
    local $x = 20; # 'local' provides a temporary value to the global x
    print_x();
}

main();
```

Lab Tasks

Task 1

Run the following python code, and explain the output.

```python
a = 5

def f():
  print(a)

def g():
  a = 10
  f()

g()
```

Task 2

Convert the Python code from **Task 1** into Perl syntax. You will perform a two-part experiment by modifying the variable declaration inside the function g().

Case A: Using `'my'` Keyword

Case B: Using `'local'` Keyword

Run the code and record the output. Explain the output for two cases.

Task 3

Complete the Perl code below by filling in the blanks with either `'my'` (Lexical) or `'local'` (Dynamic). For each case, run the code in Perl, and explain the output.

```perl
$x = "Global";

sub level_3 {
    print "Level 3 sees x as: $x\n";
}

sub level_2 {
    # TO DO
    ______ $x = "Level 2 Value";
    level_3();
}
```

```
sub level_1 {
    # TO DO
    ______ $x = "Level 1 Value";
    level_2();
}

level_1();
```

Cases you have to explain:

Case	Level 1 Key word	Level 2 Keyword
A	`my`	`my`
B	`local`	`my`
C	`my`	`local`
D	`local`	`local`

Submission

Create a single PDF file containing all your source codes, execution screenshots, and explanations. Name the file CO523_Lab04_EXXYYY.pdf, where EXXYYY represents your E-number. Ensure all code is properly formatted and readable before submission.

Upload your answer sheet to the FEeLS by the given deadline.