Bilkent University

Department of Computer Engineering

CS 315

Programming Language

Homework 1

Static & Dynamic Scoping

02.04.2018

Fuad Aghazada: 21503691

Section 1

Instructor:

- Halil Altay Güvenir

Table of contents

Example Program in 5 different languages

1. C language

1.1 Code

```
/***
   Analyzing Static scoping on C language

   @author: Fuad Aghazada
   @date:   29/03/2018
*/

#include<stdio.h>

//Test variable for the experiment
int testVariable = 7;

/*
  Function goo -- prints the testVariable (global) without declaring a local one.
*/
int goo()
{
   printf("In goo: %d \n", testVariable);    // static scope: in parent testVariable = 7

   return testVariable;
}

/*
  Function foo -- prints the testVariable by declaring a new local variable.
*/

int foo()
{
   int testVariable = 20;

   printf("In foo: %d \n", testVariable);          // static scoping: nearest variable—20

   return goo();
}
```

```
/*
  Function bar -- prints the testVariable by declaring a new local one.
*/

int bar()
{
   int testVariable = 19;

   printf("In bar: %d \n", testVariable);                    // static scoping: nearest variable—19

   return foo();
}

// main method for calling the function bar
int main()
{
   printf("Before calling function bar testVariable: %d \n", testVariable);

   bar();                                                    // calling the parent function bar

   printf("After calling function bar testVariable: %d \n", testVariable);
}
```

1.2 Output

Before calling function bar testVariable: 7
In bar: 19
In foo: 20
In goo: 7
After calling function bar testVariable: 7

1.3 Explanation

C is one of the most popular imperative languages, which has a static scoping rule. According to the rule of static scoping, scope of a variable can be called within a block of code, in which this variable is defined. In case of my example program the execution goes on this rule so that ‘goo’ returns the value of the test variable which is called by another function foo, in which another variable with the same name is defined (it is the nearest variable to the call of function), and in the same manner function foo is called by the function bar - it is the function called in the main method to see the result of the print statements on the console.
C does not have dynamic scoping.

2. JavaScript

2.1 Code

```
/**
   Analyzing Static scoping on JavaScript language

   @author: Fuad Aghazada
   @date:   29/03/2018
*/

//Test variable for the experiment
var testVariable = 7;

// function bar with local variable (value 19)
function bar()
{
   var testVariable = 19;

   console.log("In bar:" + testVariable);                  // static scoping at nearest - 19

   return foo();
}

// function bar with local variable (value 20)
function foo()
{
   var testVariable = 20;

   console.log("In foo:" + testVariable);                  // static scoping at nearest - 20

   return goo();
}

// function bar without local variable
function goo()
{
   console.log("In goo:" + testVariable);                  //static scoping in parent block - 7

   return testVariable;
}

console.log("Before calling function bar testVariable: " + testVariable);
bar();                                                     // calling the parent function
console.log("After calling function bar testVariable: " + testVariable);
```

2.2 Output

Before calling function bar testVariable: 7
In bar: 19
In foo: 20
In goo: 7
After calling function bar testVariable: 7

Note: JavaScript runs in client-side, so that the output be obtained from the developer console of the browsers.

2.3 Explanation

JavaScript is a script language with the rule of static scoping. In the same manner the same output with the previous language - C, will be displayed.
JavaScript also does not have dynamic scoping.

---

3. Python

3.1 Code

```
# Analyzing Static scoping on Python language

# @author: Fuad Aghazada
# @date:   29/03/2018

# Test variable for the experiment
testVariable = 7

def bar():
    testVariable = 19

    print("In bar: " + str(testVariable) + "\n")            # static scoping nearest - 19

    return foo()

def foo():
    testVariable = 20

    print("In foo: " + str(testVariable) + "\n")            # static scoping nearest - 20
```

```
    return goo()

def goo():
    print("In goo: " + str(testVariable) + "\n")            # static scoping parent block - 7
    return testVariable


print("Before calling function bar testVariable: " + str(testVariable) + "\n")
bar()                                                       # calling the parent function
print("After calling function bar testVariable: " + str(testVariable) + "\n")
```

3.2 Output

Before calling function bar testVariable: 7
In bar: 19
In foo: 20
In goo: 7
After calling function bar testVariable: 7

Note: The program was written in python 3.

3.3 Explanation

Python language is also one of the imperative languages with the rule of static scoping. The same output will be obtained in the same manner of execution. Python does not also have dynamic rule of scoping.

4. Perl

4.1 Code

4.1.1 Static scoping

```
# Analyzing Static scoping on Perl language

# @author: Fuad Aghazada
# @date:   29/03/2018

# Test variable for the experiment
$testVariable = 7;

# Function bar - printing the value of my variable
sub bar
{
   my $testVariable = 19;              # my keyword is used, so it Perl will use static scoping

   print "In bar: $testVariable"."\n";   # print statement for indicating value for $testVariable in
foo function

   return foo();                       # returning
}

# Function foo - print the value of $testVariable
sub foo
{
   my $testVariable = 20;              # my keyword is used, so it Perl will use static scoping

   print "In foo: $testVariable"."\n";   # print statement for indicating value for $testVariable in
foo function

   return goo();
}

# Function goo - print the value of $testVariable
sub goo
{
   print "In goo: $testVariable"."\n";   # print statement for indicating value for $testVariable in
foo function

   return $testVariable;
}

print "Before calling function bar testVariable: $testVariable "."\n";  # before bar()
```

```
bar();                                          # calling function bar

print "After calling function bar testVarible: $testVariable"."\n";    # after bar()
```

4.1.2 Dynamic scoping

```
# Analyzing Dynamic scoping on Perl language

# @author: Fuad Aghazada
# @date:   29/03/2018

# Test variable for the experiment
$testVariable = 7;

# Function bar - printing the value of local variable
sub bar
{
   local $testVariable = 19;              # local keyword is used, so it Perl will use dynamic
scoping

   print "In bar: $testVariable"."\n";   # print statement for indicating value for $testVariable in
foo function

   return foo();                          # returning
}

# Function foo - print the value of $testVariable
sub foo
{
   local $testVariable = 20;             # local keyword is used, so it Perl will use dynamic
scoping

   print "In foo: $testVariable"."\n";   # print statement for indicating value for $testVariable in
foo function

   return goo();
}

# Function goo - print the value of $testVariable
sub goo
{
   print "In goo: $testVariable"."\n";   # print statement for indicating value for $testVariable in
foo function

   return $testVariable;
```

```
}

print "Before calling function bar testVariable: $testVariable "."\n";  # before bar()

bar(); # calling function bar

print "After calling function bar testVarible: $testVariable"."\n";    # after bar()
```

4.2 Output

4.2.1 Static scoping

Before calling function bar testVariable: 7
In bar: 19
In foo: 20
In goo: 7
After calling function bar testVariable: 7

4.2.2 Dynamic scoping

Before calling function bar testVariable: 7
In bar: 19
In foo: 20
In goo: 20
After calling function bar testVariable: 7

4.3 Explanation

Perl is one of the imperative languages which has both the rules of static and dynamic scoping. In Perl in order to use the static scoping variables with ‘my’ keyword are used. The case of the static scoping is the same with the other imperative languages such as C, Python and etc. However, if ‘local’ keyword is used in Perl, it means that the rule of dynamic scoping is used. In case of dynamic scoping the variable is accessed over the function which is called. In my example, in function goo the test variable is 20, which is according to the call of function foo. In conclusion, Perl has both static and dynamic scoping.

5. PHP

5.1 Code

```
/***
    Analyzing Static scoping on PHP language

    @author: Fuad Aghazada
    @date:   29/03/2018
 */


 // Test Variable
 $testVariable = 7;

 function bar()
 {
    $testVariable = 19;

    echo nl2br("In bar: $testVariable \r\n");          // static scoping nearest - 19

    return foo();
 }

 function foo()
 {
    $testVariable = 20;

    echo nl2br("In foo: $testVariable \r\n");          // static scoping nearest - 20

    return goo();
 }

 function goo()
 {
    global $testVariable;                          // in order to access – need global in the block

    echo nl2br("In goo: $testVariable \r\n");          // static scoping parent block - 7

    return $testVariable;
 }

 echo nl2br("Before calling function bar testVariable: $testVariable \r\n");

 bar();                                         // calling the parent function
```

```
echo nl2br("After calling function bar testVariable: $testVariable \r\n");
```

5.2 Output

Before calling function bar testVariable: 7
In bar: 19
In foo: 20
In goo: 7
After calling function bar testVariable: 7

Note: Since PHP is a server side language, I compiled and run it through a local server created by XAMPP. (at Dijkstra server too)

5.3 Explanation

PHP is also a language with the rule of static scoping. The same output is obtained from the example program written in PHP. The only difference in the code is that in order to access a variable in a function block from outside of the block, I needed to put ‘global’ in front of the variable, because without that the variable is not global and cannot be access from the function.
PHP also does not have the rule of dynamic scoping.

6. Advantages and limitations

Analyzing the scoping rules for these 5 languages, I concluded that only one of them - Perl has the rule of dynamic scoping together with the static scoping. Although having variety of features in a language increases the flexibility of the language, it creates also some difficulties. For example, it is obviously straightforward to read and predict the output of the code in the examples of static scoping so having static scoping increases the readability of the program. Nevertheless, in the case of dynamic scoping, it is getting harder to accustom and read the program. In addition, for dynamic scoping it would also be unreliable program, since the variables are visible to the subprograms which they called and they can be misused. As a result we may not be able to get the correct output in case we do not know what we are writing.