Universität Stuttgart

Programming Paradigms Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart, Winter 2019/2020

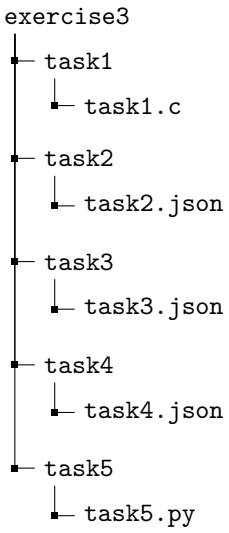
Exercise 3: Names, Scopes, and Bindings

(Deadline for uploading solutions: Nov 24, 2019, 11:59pm Stuttgart time)

The materials provided for this homework are:

- a pdf file with the text of the homework (this);
- a zip file with **the folder structure and the templates that must be used for the submission**.

The folder structure is:



The submission must be compressed in a zip file (**not rar** or other formats) using the given folder structure. The name of folders and files must not be changed or moved, otherwise the homework will not be evaluated. Late submissions will be not accepted.

There are five tasks, which contribute a specific percentage to the overall points for this exercise.

1 Task I (20% of total points of the exercise)

You get a C program with a scope-related bug. The code is in the file *exercise3/task1/task1.c*

You have to fix the bug by moving some lines of code, without adding new code or deleting code. For example, you can move variable declarations, function calls, if statements, etc., but you cannot add new variables, new function calls, etc. The correct behavior of the program is the computation of the Fibonacci numbers with $n = 5$, so the final output should be: *"Fibonacci of 5: 0 1 1 2 3"*.

Your solution has to compile and work to be accepted. When grading this task, we will use the *gcc 7.4.0* compiler. You have to write your solution into the file *exercise3/task1/task1.c*

2 Task II (20% of total points of the exercise)

This task is about understanding the function stack of the following C program:

Task2.c

```
1  #include <stdio.h>
2
3  void a();
4  void b(int x);
5  int c(int x);
6  void d(int x);
7  void e(int x);
8
9  int v = 0;
10
11 int main() {
12     a();
13     return 0;
14 }
15
16 void a (){
17     int x = 5;
18     if(1)
19         v = 2;
20     b(5*2+v);
21 }
22
23 void b(int x){
24     int f = 3;
25     int z = c(x / f + f);
26     d(z);
27 }
28
29 int c(int x){
30     int i = 0;
31     int y = 4;
32     return i + y + x;
33
34 }
35
36 void d(int x){
37     int y = v;
38     printf("%d\n", y); // <== Write the stack when the program arrives here
39 }
40
41 void e(int x){
42     int z = v;
43 }
```

The goal of this exercise is to write into *exercise3/task2/task2.json* the function stack, including all local variables and their values stored on the stack, when the program arrives at line 38.

To illustrate the task and the expected format for the function stack, consider the following example:

Example.c

```c
#include <stdio.h>

void foo1();
void foo2();

int main() {
    foo1();
    return 0;
}

void foo1 (){
    int a = 5;
    int b = 1;
    foo2();
}

void foo2(){
    int c = 3;
    // Write the stack when
    // the program arrives here
}
```

Example.json

```json
{
  "foo2": {
    "c": [
      3
    ]
  },
  "foo1": {
    "a": [
      5
    ],
    "b": [
      1
    ]
  },
  "main": {},
}
```

In the JSON file, the functions are ordered as on the call stack, i.e., the most recently called function on the top.

3 Task III (20% of total points of the exercise)

The goal of this task is to understand the difference between static and dynamic scoping. Two pseudo-codes are provided.

3.1 Code 3.1

Code3.1

```
var x = 0
var y = 2

foo1(){
    x = x + 1
}

foo2(){
    x = x + 1
    foo1()
}

if(y > 0) {
    var x //it is a new local variable only for this block
    foo2()
}

print(x)
```

3.2 Code 3.2

code3.2

```
var x

set_x(n){
    x = n
}

print_x(){
    print(x)
}

first(){
    set_x(1)
}

second(){
    var x //it is a new local variable only for this block
    set_x(2)
}

set_x(0)
first()

second()
print_x()
```

The aim of this task is to understand which value of x the program prints in a programming languages with static scoping and dynamic scoping, respectively. Your answer has to be written into *exercise3/task3/task3.json*, using the provided JSON file:

codeC.json

```
{
  "code3.1": {
    "static": [
      0
    ],
    "dynamic": [
      1
    ]
  },
  "code3.2": {
    "static": [
      2
    ],
    "dynamic": [
      3
    ]
  }
}
```

where instead of 0,1,2,3 you have to put the correct values of the printed x.

4 Task IV (20% of total points of the exercise)

In this task, some Java code is given. There are multiple aliases for the value that the field *int[] array_numbers* of the object *mClass* refers to.

Me.java

```
1  public class Me {
2
3      public static void main(String[] args)
4      {
5          int numbers[] = {1,2};
6          MyClass mClass = new MyClass();
7          mClass.setNumber(5);
8          mClass.setArray(numbers);
9
10         int n = mClass.getNumber();
11         int new_array[] = mClass.getArray();
12         int int_array[] = new_array;
13
14         System.out.println(mClass.getNumber());
15         System.out.println(mClass.getArray()[0] + " " +mClass.getArray()[1]);
16
17         mClass.setOtherField(new_array);
18
19         foo(mClass);
20
21         n = 10;
22         new_array[0] = 3;
23         new_array[1] = 4;
24
25         System.out.println(mClass.getNumber());
26         System.out.println(mClass.getArray()[0] + " " +mClass.getArray()[1]);
27
28         int_array[0] = 99;
29         int_array[1] = 98;
30
31         System.out.println(mClass.getNumber());
32     }
33
34     public static void foo(MyClass c)
35     {
36         int a[] = {-1,-2};
37         c.setArray(a);
38     }
39 }
```

MyClass.java

```
1  public class MyClass {
2      private int single_number;
3      private int[] array_numbers;
4      private int[] other_field;
5
6      public void setNumber(int single_number){
7          this.single_number = single_number;
8      }
9      public void setArray(int array_numbers[]){
10         this.array_numbers = array_numbers;
11     }
12     public void setotherField(int array_numbers[]){
13         this.other_field = array_numbers;
```

```
14      }
15      public int getNumber(){
16          return single_number;
17      }
18      public int[] getArray(){
19          return array_numbers;
20      }
21  }
```

The goal of this task is to find all the aliases of the value referred to by *int[] array_numbers* of object *mClass*, and to write them into *exercise3/task4/task4.json*.
The structure of the JSON file is provided in the file *exercise3/task4/task4.json*:

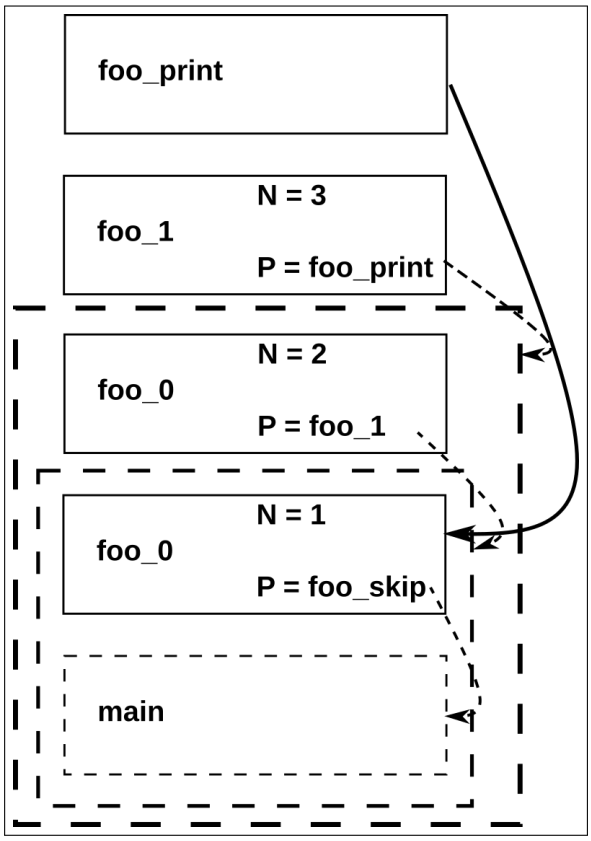
task4.json

```
1  [
2    [
3      "Me",
4      "main",
5      "mClass",
6      "array_numbers"
7    ],
8    [
9      "Me",
10     "main",
11     "array"
12   ],
13   [
14     "MyClass",
15     "otherField"
16   ]
17 ]
```

where for each alias, you have to put the file name (ex. Me), the method name (ex. main, but for example not in the file MyClass), then the variable (if it an object an other field is required for the variable).

5 Task V (20% of total points of the exercise)

In the last task the referencing environment topic is covered, in particular deep binding using Python. The file *exercise3/task5/task5.py* contains an incomplete Python program. Figure 1 represents the conceptual view of the run-time stack of the solution of this task.



task5.py

```
1  def foo_0(N, P):
2      def foo_print():
3          print(N)
4      def foo_1():
5          #insert something
6
7      #body of foo_0
8      if N > 1:
9          #insert something
10     else:
11         #insert something
12
13 def foo_skip():
14     pass #do nothing
15
16 foo_0(1, foo_skip) # main program
```

Figure 1: The conceptual view of the run-time stack.

where referencing environments captured in closures are shown as dashed boxes and arrows.

The goal of this task is to complete the code by adding code wherever there is the comment *#insert something*, so that we get the behavior shown in Figure 1. If the program is correct, the output of *foo_print* is the value 1. You have to write your solution into the file *exercise3/task5/task5.py*