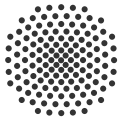

University of Stuttgart

Programming Paradigms Prof. Dr. Michael Pradel

University of Stuttgart, Summer 2020

Exercise 3: Names, Scopes, and Bindings

(Deadline for uploading solutions: June 5, 2020, 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:

```
exercise3
├── task1
│   └── task1.cpp
├── task2
│   └── task2.csv
├── task3
│   └── task3.csv
├── task4
│   └── task4.csv
└── task5
    └── task5.csv
```

The submission must be compressed in a zip file using the given folder structure. The name of folders and files must not be changed or moved, otherwise the homework will not be evaluated.

General tips for making sure your submission is graded as you expect:

- Use only the zip format for the archive (**not rar**, 7z, etc.) ;
- Your zip file should contain exactly one top-level directory *"exercise3/"*, as in the zip file provided by us.
- Do not rename files or folders, simply open the files provided and put your solutions;

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)

This task focuses on scopes, variable declarations and function invocations. You are provided with a buggy implementation of a sorting algorithm. The code is in the file *exercise3/task1/task1.cpp*

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.

Evaluation Criteria: You are expected to submit a compilable and working sorting algorithm. The task will be compared against the correctly sorted result of multiple inputs. You have to write your solution into the file *exercise3/task1/task1.cpp*. When grading this task, we will use the *gcc 7.5.0* compiler.

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

This task is about understanding the function stack of the following C program and to write into *exercise3/task2/task2.csv*. For the task you are required to include all local variables and their values stored on the stack, when the program arrives at a particular line.
Note: All functions are to be ordered as on the call stack, i.e., the most recently called function on the top.
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.csv

```
Function_Name,Variable,Value
foo2,c,3
foo1,b,1
foo1,a,5
main,,
```

Task: You are expected to submit a well-formed csv file containing function stack, when the program arrives at line 39. You have to submit your answers in *exercise3/task2/task2.csv*

Evaluation Criteria: Your answer will be evaluated on the correct order of the function being called and the values of the variables.

Task2.c

```c
#include <stdio.h>

void a();
void b(int x);
void c(int x);
int d(int x);
int e(int x);

int v = 0;

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

void a (){
    int x = 5;
    if(x<6)
        v=e(x);
    c(5*2+v);
}

void c(int x){
    int f = 3;
    int z = d(e(x / f + f));
    b(z);

}

int e(int x){
    int i = 0;
    int y = 4;
    return i + y + x;

}

void b(int x){
    int y = v;
    printf("%d\n", y); // <== Write the stack when the program arrives here
}

int d(int x){
        int z=v;
        for(int i=0;i<x;i++){
        z=z+i;
    }
    return z;
}
```

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. You are provided with two pseudo-codes. The aim of this task is to understand which value of **x** the program prints in a programming language with static scoping and dynamic scoping, respectively. Your answers have to be written into *exercise3/task3/task3.csv*, using the provided csv file:

3.1 Code 3.1

Code3.1

```
1  var x = 9
2  var y = 2
3
4  foo2(){
5      x = x + 1
6  }
7
8  foo1(){
9      x = x/y
10     foo2()
11 }
12
13 if(y > 0) {
14     var x //it is a new local variable only for this block
15     foo1()
16 }
17
18 print(x)
```

3.2 Code 3.2

code3.2

```
1  var x
2
3  set_x(n){
4      x = n+1
5  }
6
7  print_x(){
8      print(x)
9  }
10
11 second(){
12     set_x(1)
13 }
14
15 first(){
16     var x //it is a new local variable only for this block
17     set_x(2)
18 }
19
20 set_x(0)
21 first()
22
23 second()
24 print_x()
```

task3.csv

```
Code_Name,Scoping,Value
code3.1,static,value-1
code3.1,dynamic,value-2
code3.2,static,value-3
code3.2,dynamic,value-4
```

Note: You have to substitute your answers in place holders value-1, value-2, value-3, value-4. Also consider all variables as Integers in this code.
Evaluation Criteria: Your answer will be evaluated on the correct values of the printed x.

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

The goal of this task is to understand aliases. You are provided with a Java code containing usage of a class called *"CustomObject"* and multiple functions (e.g. create, copy etc). The *Main* class invokes these functions to create aliases of the object.
Task: In this task, you are required to understand the given code and answer whether two objects *may* be aliases of each other. You are provided with the list of variables and the line they appear in. You need to answer **Y** or **N** to indicate whether a pair of variables may alias. Submit your answers in the provided file *exercise3/task4/task4.csv*.

To illustrate the task and the expected format for the file consider the following example code, sample variable list and csv file :

variable list to consider

```
1 variable f at line 2
2 variable g at line 3
3 parameter f at line 10
4 variable h at line 6
```

sample code

```
1  main() {
2    Foo f = Foo();
3    Foo g = f;
4    Foo h;
5    for (int i = 0; i< 2; i++) {
6          h = a(g, i);
7    }
8  }
9
10 Foo a(Foo f, int i) {
11   if (i==0)
12     return new Foo();
13   else
14         f;
15 }
```

Example.csv

```
Index,1,2,3,4
1    ,Y,Y,Y,Y
2    , ,Y,N,Y
3    , , ,Y,Y
4    , , , ,Y
```

Note: In the csv file, variables are replaced with the index they are listed with for readability (e.g., variable f at line 2 is index 1). Furthermore, you only need to fill in the upper half of the file, since the lower half has the repeated pair.
Evaluation Criteria: Your answers will be evaluated on the correct identification of aliases in the csv.

Main.java

```
1  public class Main {
2   public static void main(String[] args) {
3
4   CustomObject cObj1 = new CustomObject(1);
5   CustomObject cObj2 = new CustomObject(2);
6   CustomObject cObj3 = new CustomObject(3);
7
8   List<String> messageList = cObj3.getMessages();
9   CustomObject cObj4 = create(cObj1);
10  cObj2.setMessages(copyMessages(messageList));
11
12  cObj3 = copy(cObj4);
13
14  messageList.clear();
15  for (String message : cObj4.getMessages()) {
16    messageList.add(message);
17  }
18
19  CustomObject customObjArray[] = new CustomObject[5];
20  for (int i = 0; i < 5; i++) {
21   switch (i) {
22    case 0:
23    case 3:
24     customObjArray[i] = new CustomObject(0);
25    break;
26    case 1:
27    case 2:
28     customObjArray[i] = create(cObj4);
29    break;
30    case 4:
31     customObjArray[i] = copy(cObj3);
32    break;
33    }
34   }
35  }
36
37  public static CustomObject copy(CustomObject c) {
38    CustomObject newCustomObject = new CustomObject(c.id);
39    for (String message : c.messages) {
40      newCustomObject.addMessage(message);
41    }
42    return newCustomObject;
43  }
44
45  public static List<String> copyMessages(List<String> messages) {
46    List<String> copyList = new ArrayList<String>();
47    List<String> newList = new ArrayList<String>();
48    for (String message : messages) {
49      newList.add(message);
50    }
51    return copyList;
52  }
53
54  public static CustomObject create(CustomObject original) {
55    CustomObject newObj = new CustomObject();
56    newObj.id = original.id;
57    newObj.messages = original.messages;
58    return newObj;
59   }
60 }
```

CustomObject.java

```
1  public class CustomObject{
2    int id;
3    List<String> messages;
4
5    public int getId() {
6      return id;
7    }
8
9    public void setId(int id) {
10     this.id = id;
11   }
12
13   public CustomObject() {
14   }
15
16   public CustomObject(int id) {
17     this.id=  id;
18     this.messages= new ArrayList<String>();
19   }
20
21   public List<String> getMessages() {
22     return messages;
23   }
24
25   public void addMessage(String customStmt) {
26     this.messages.add(customStmt);
27   }
28
29   public void setMessages(List<String> messages) {
30     this.messages = messages;
31   }
32
33 }
```

variable list for Task IV

```
1  variable cObj1 at line 5
2  variable cObj2 at line 6
3  variable cObj3 at line 7
4  property cObj3.messages at line 9
5  variable messageList at line 9
6  variable cObj4 at line 10
7  property cObj2.messages  at line 11
8  variable customObjArray[0] at line 25
9  variable customObjArray[1] at line 29
10 variable customObjArray[2] at line 29
11 variable customObjArray[3] at line 25
12 variable customObjArray[4] at line 32
13 parameter c at line 38
14 parameter messages at line 43
15 variable original at line 55
```

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

In the last task the referencing environment topic is covered, in particular deep binding and shallow binding. In this task you are provided with a pseudo-code where functions set_x and print_x are arguments to function run. The pseudo-code follows Python like syntax and semantics. Also assume, the code follows Dynamic scoping.

Task: You are required to list the output of print_x called during each iterations for shallow and deep binding. Your answers have to be written into *exercise3/task5/task5.csv*, using the provided csv file.

Note: During each iteration, print_x is called twice. First at line 13 or line 15 depending upon the condition, output of this invocation is to be written in *output1*. Where as, output of print_x at line 20 is to be written under *output2* in the csv file.
Evaluation Criteria: Your answer will be evaluated on the correct values for shallow and deep binding at each print statement.

task5

```
1  x = 0
2  function set_x(n):
3    x=n
4  function print_x():
5    print(x)
6  function run(s,p,n):
7    x= 5
8    if n in [1,3]:
9      set_x(n)
10   else:
11     s(n)
12   if n in [1,2]:
13     print_x() # output 1
14   else:
15     p() #output 1
16 #main
17 for (i = 1; i < 5; i++)
18   set_x(0)
19   run(set_x,print_x,i)
20   print_x() #output 2
```

You have to write your solution into the file *exercise3/task5/task5.csv*

task5.csv

```
Iteration,Binding,output1,output2
1,deep,,
1,shallow,,
2,deep,,
2,shallow,,
3,deep,,
3,shallow,,
4,deep,,
4,shallow,,
```