Software Quality Assurance (WS20/21)
Problem Set 5

Problem 1: Data Flow Anomaly Analysis

A software company develops software packages for commercial animal housing. A particular function, which is implemented in the C programming language, computes the daily amount of feed for different animal species depending on their individual weight.

Until now, this function was only part of a software package for farms and worked failure-free since years. Recently, it is also included in a software package for zoological gardens and it produces wrong output in some cases. By performing a data flow analysis, the faults should be revealed.

```
/* Own data type for enumeration of animal species */
typedef enum {COW, HORSE, PIG, ELEPHANT} Animal_A;

/* Function for determining the daily amount of feed depending
 * on the animal species and the individual weight
 */
01 float feedamount(Animal_A species, float weight)
02 {
03   float amount, factor;
04   switch (species)
05   {
06     case COW:
07     {
08       factor = 0.05;
09       break;
10     }
11     case HORSE:
12     {
13       factor = 0.1;
14       break;
15     }
16     case PIG:
17     {
18       factor = 0.02;
19       break;
20     }
21   } // end switch
22   amount = factor * weight;
23   return amount;
24 } // end feedamount
```

a) What mistakes were performed and how would the consequences have been avoided?

-

No default case, not all cases taken into account -> in case of ELEPHANT factor undefined

Factor not initialized

b) Perform a data flow anomaly analysis for the operation feedamount.

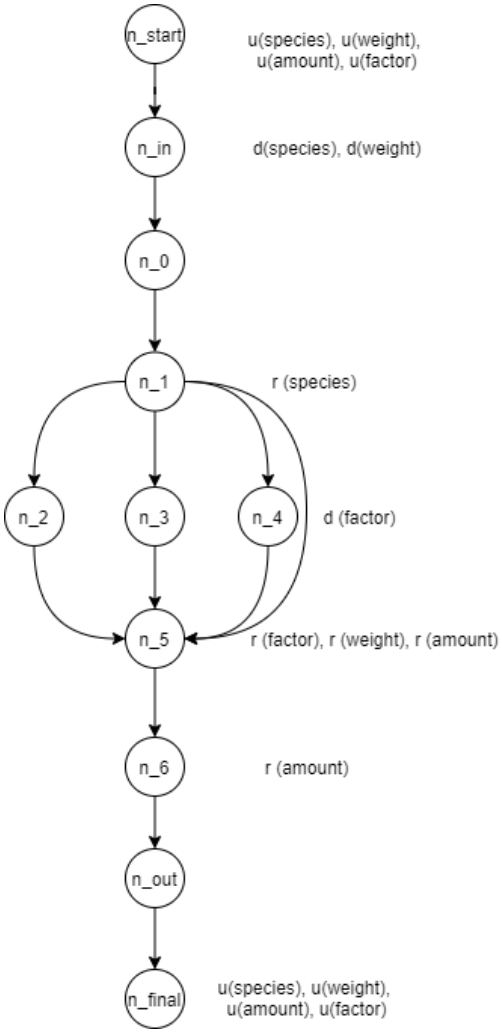
-

path 1	n_start	n_in	n_0	n_1	n_2	n_5	n_6	n_out	n_final
species	u	d		r					u
weight	u	d				r			u
amount	u		u			d	r		u
factor	u		u		d	r			u

path 2	n_start	n_in	n_0	n_1	n_3	n_5	n_6	n_out	n_final
species	u	d		r					u
weight	u	d				r			u
amount	u		u			d	r		u
factor	u		u		d	r			u

path 3	n_start	n_in	n_0	n_1	n_4	n_5	n_6	n_out	n_final
species	u	d		r					u
weight	u	d				r			u
amount	u		u			d	r		u
factor	u		u		d	r			u

path 4	n_start	n_in	n_0	n_1	n_5	n_6	n_out	n_final
species	u	d		r				u
weight	u	d			r			u
amount	u		u		d	r		u
factor	u		*u*		*r*			u



The data flows of the paths 1 to 3 are equivalent and free of data flow anomalies. In path 4, an ur-anomaly can be found for the variable factor.

Problem 2: Data Flow Anomaly Analysis

Consider the following Java implementation of the operation ALL_POSITIVE which checks whether all elements of a one-dimensional array are positive. As parameters, the field and its length are given.

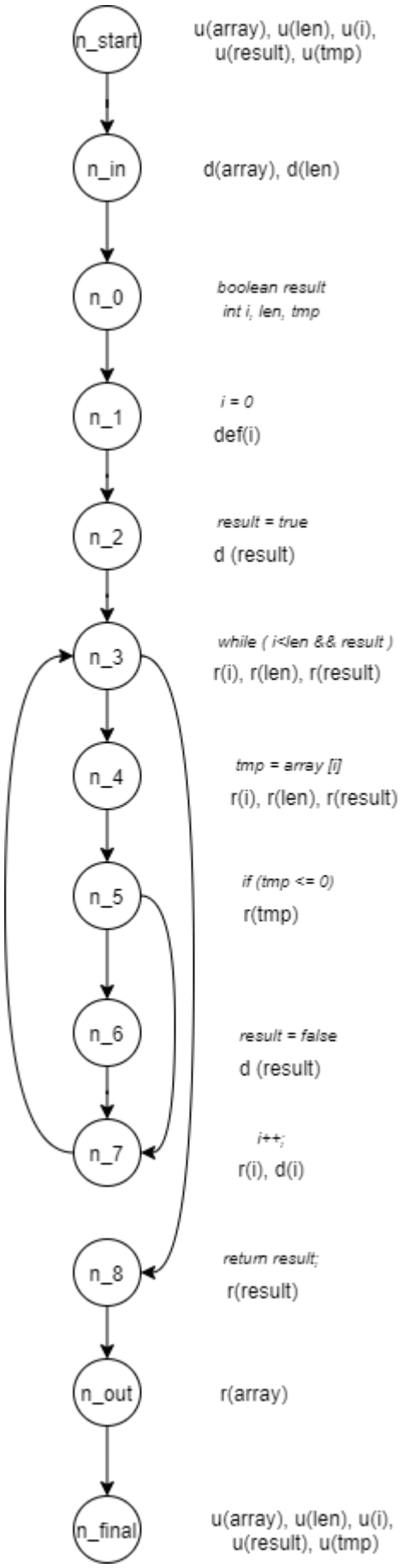
```
01 boolean ALL_POSITIVE(int[] array,int len) {
02   boolean result;
03   int i,tmp;
04   i=0;
05   result=true;
06   while (i<len&&result) {
07     tmp=array[i];
08     if (tmp<=0)
09       result=false;
10     i++;
11   }
12   return result;
13 }
```

Perform a data flow anomaly analysis for the operation ALL_POSITIVE.

-

n = 0	n_start	n_in	n_0	n_1	n_2	n_3	n_8	n_out	n_final
array	u	d						r	u
len	u	d				r			u
i			u	d		r			u
result			u		d	r	r		u
tmp			u						u

array: udru
len: udru
i: udru
result: udrru
tmp: uu



n = 1	n_start	n_in	n_0	n_1	n_2	n_3	n_4	n_5	n_7	n_3	n_8	n_out	n_final
array	u	d					r					r	u
len	u	d				r				r			u
i			u	d		r	r		r, d	r			u
result			u		d	r				r	r		u
tmp			u				d	r					u

array: ud (r) ru
len: udr (r) u
i: udr (rrdr) u
result: udr (r) ru // node n5 - n7
tmp: u (dr) u

array: ud (r) ru
len: udr (r) u
i: udr (rrdr) u
result: udr ((d)^1 r) ru // node n5 – n6 - n7
tmp: u (dr) u

n =2	n_start	n_in	n_0	n_1	n_2	n_3	n_4	n_5	n_7	n_3	n_4	n_5	n_6	n_7	n_3	n_8	n_out	n_final
array	u	d					r				r						r	u
len	u	d				r				r					r			u
i			u	d		r	r		r, d	r	r			r,d	r			u
result			u		d	r				r			d		r	r		u
tmp			u				d	r			d	r						u

array: ud (r) (r) ru
len: udr (r) (r) u
i: udr (rrdr) (rrdr) u
result: udr (r) (r) ru // node n5 - n7
tmp: u (dr) (dr) u

array: ud (r) (r) ru
len: udr (r) (r) u
i: udr (rrdr) (rrdr) u
result: udr (r) ((d) r) ru // node n5 – n6 - n7
tmp: u (dr) (dr) u

Expression that represents the data flow
array: ud (r)*n* ru
len: udr (r)*n* u
i: udr (rrdr)*n* u
result: udr ((d) *k* r)*n* ru
tmp: u (dr)*n* u

So, No data flow anomalies

Problem 3: Slicing

Create static backward slices for the last occurrence of variables: ***result, mode and this.mode***.

```
01 public class Switch {
02   private boolean mode;
03   public Switch() {
04     init();
05   }
06   private void init() {
07     mode=true;
08   }
09   public boolean toggle(boolean mode) {
10     boolean result;
11     if((this.mode&&mode)||(!this.mode&&!mode))
12       result=true;
13     else
14       result=false;
15     if (this.mode)
16       this.mode=!mode;
17     else
18       mode=mode;
19     return result;
20   }
21 }
```

-

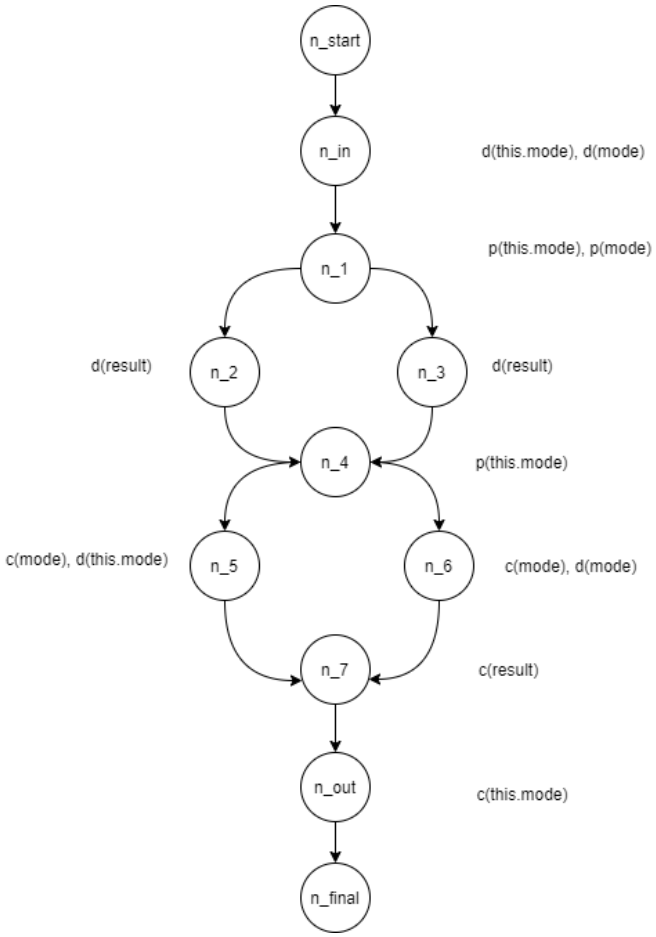


Figure: Control flow