Task2 - SCSR

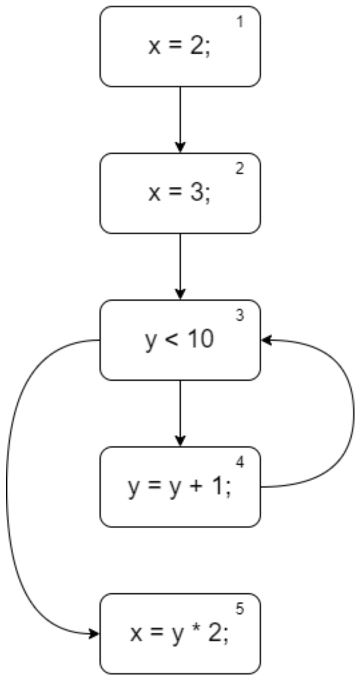
Andrea Munarin (879607), Simone Jovon (882028)

February 2023

1 Exercise 1: reaching definitions

```
x = 2;
x = 3;
while (y < 10)
    y = y + 1;
x = y * 2;
```

SOLUTION
1. Build the control flow graph:



2. Compute the reaching definition analysis, i.e., compute the in-set and the out-set of each node of the graph

We remember that given the direction of analysis forward, and the confluence operator the union, we define how we populate the sets:

- `gen[B]` = {definitions that appear in B and reach the end of B}
- `kill[B]` = {all definitions that never reach the end of B}
- `out[B]` = `gen[B]` $\cup$ (`in[B]-kill[B]`)
- `in[P]` = $\cup$ `out[Q]` over the predecessors Q of P

We initialize each block `out[B]` = `gen[B]` So we can define and then provide for each Basic Block the RD_{in} and the RD_{out}:

- $RD_{in}(p) = \emptyset$ if p is the initial point in the graph
- $RD_{in}(p) = \cup\{RD_{out}(q)|$ *there is an arrow form q to p in the CFG*$\}$
- $RD_{out}(p) = gen_{RD}(p) \cup (RD_{in}(p) \setminus kill_{RD}(p))$

$RD_{in}(1)$ = {(x,?),(y,?)}
$RD_{out}(1)$ = {(x,1),(y,?)}

$RD_{in}(2)$ = {(x,1),(y,?)}
$RD_{out}(2)$ = {(x,2),(y,?)} (x,1) killed by block 2

$RD_{in}(3) = RD_{out}(2) \cup RD_{out}(4)$ = {(x,2),(y,?)(y,4)}
$RD_{out}(3)$ = {(x,2),(y,?)(y,4)}

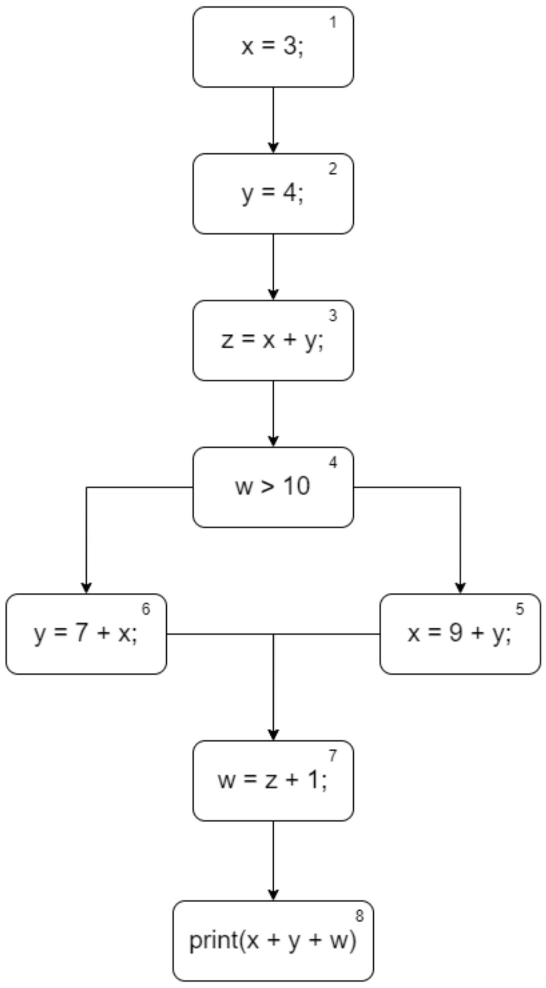
$RD_{in}(4)$ = {(x,2),(y,?)(y,4)}
$RD_{out}(4)$ = {(x,2),(y,4)}

$RD_{in}(5)$ = {(x,2),(y,?)(y,4)}
$RD_{out}(5)$ = {(x,5),(y,?)(y,4)}

2 Exercise 2: constant folding

```
x = 3
y = 4
z = x + y
if (w > 10)
    x=9+y
else
    y=7+x
w=z+1
print(x+y+w)
```

SOLUTION
1. Build the control flow graph:



2. Compute the reaching definition analysis, i.e., compute the in-set and the out-set of each node of the graph

$RD_{in}(1)$ = {(x,?),(y,?),(z,?),(w,?)}
$RD_{out}(1)$ = {(x,1),(y,?),(z,?),(w,?)}

$RD_{in}(2)$ = {(x,1),(y,?),(z,?),(w,?)}
$RD_{out}(2)$ = {(x,1),(y,2),(z,?),(w,?)}

$RD_{in}(3)$ = {(x,1),(y,2),(z,?),(w,?)}
$RD_{out}(3)$ = {(x,1),(y,2),(z,3),(w,?)}

$RD_{in}(4)$ = {(x,1),(y,2),(z,3),(w,?)}
$RD_{out}(4)$ = {(x,1),(y,2),(z,3),(w,?)}

$RD_{in}(5)$ = {(x,1),(y,2),(z,3),(w,?)}
$RD_{out}(5)$ = {(x,5),(y,2),(z,3),(w,?)}

$RD_{in}(6)$ = {(x,1),(y,2),(z,3),(w,?)}
$RD_{out}(6)$ = {(x,1),(y,6),(z,3),(w,?)}

$RD_{in}(7) = RD_{out}(5) \cup RD_{out}(6)$ = {(x,1),(x,5),(y,2),(y,6),(z,3),(w,?)}
$RD_{out}(7)$ = {(x,1),(x,5),(y,2),(y,6),(z,3),(w,7)}

$RD_{in}(8)$ = {(x,1),(x,5),(y,2),(y,6),(z,3),(w,7)}
$RD_{out}(8)$ = {(x,1),(x,5),(y,2),(y,6),(z,3),(w,7)}

3. Transform the program by applying the constant folding rules, based on the reaching definition analysis

```
x = 3
y = 4
z = 7
if (w > 10)
    x=13
else
    y=10
w=8
print(x+y+8)
```

Following the steps for the transformation:

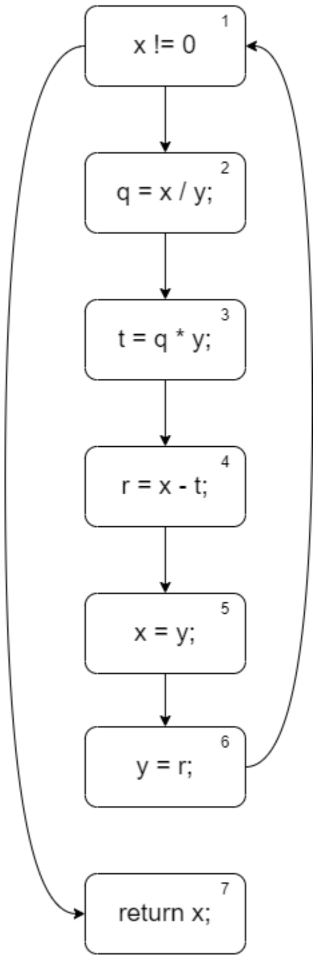
- Since $RD_{in}(3)$ = {(x,1),(y,2),(z,?),(w,?)} and x, y can have only one value, thanks to the rule 1 the statement 3 became $z = 3 + 4$;
- Since the expression $z = 3 + 4$ doesn't have free variables, thanks to the rule 2 the statement 3 became $z = 7$;
- Since $RD_{in}(6)$ = {(x,1),(y,2),(z,3),(w,?)} and x can have only one value, thanks to the rule 1 the statement 6 became $y = 7 + 3$;
- Since the expression $y = 7 + 3$ doesn't have free variables, thanks to the rule 2 the statement 6 became $y = 10$;
- Since $RD_{in}(5)$ = {(x,1),(y,2),(z,3),(w,?)} and y can have only one value, thanks to the rule 1 the statement 5 became $x = 9 + 4$;
- Since the expression $x = 9 + 4$ doesn't have free variables, thanks to the rule 2 the statement 5 became $x = 13$;
- Since $RD_{in}(7)$ = {(x,1),(x,5),(y,2),(y,6),(z,3),(w,?)} and z can have only one value, thanks to the rule 1 the statement 7 became $w = 7 + 1$;

- Since the expression $w = 7 + 1$ doesn't have free variables, thanks to the rule 2 the statement 7 became $w = 8$;
- Since $RD_{in}(8) = \{$(x,1),(x,5),(y,2),(y,6),(z,3),(w,7)$\}$ and w can have only one value, thanks to the rule 1 the statement 8 became $print(x + y + 8)$;

3 Exercise 3: liveness

```
while (x != 0){
    q = x/y
    t = q*y
    r = x-t
    x = y
    y = r
}
return x
```

SOLUTION
1. Build the control flow graph:



2. Compute the liveness analysis of the program

We remember that given the direction of analysis backward, and the confluence operator the union, we define how we populate the sets:

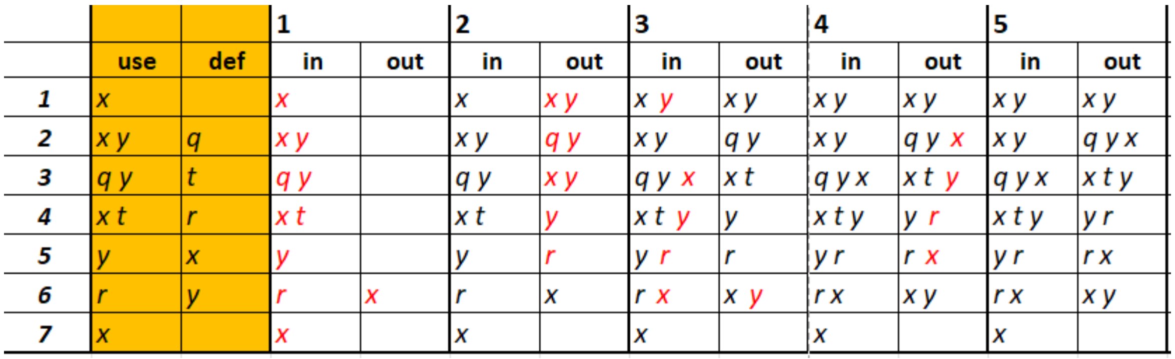
- `kill[B]` = {variables defined in B before being used}
- `gen[B]` = {variables used in B before being defined}
- `in[B]` = `use[B]` ∪ (`out[B]-def[B]`)
- `out[B]` = ∪ `in[S]` over the successor S of B

We initialize each block `in[B]` = ∅ So we can define and then provide for each Basic Block the LV_{entry} and the LV_{exit}:

- $LV_{exit}(p) = \emptyset$ if p is a final point in the graph

- $LV_{exit}(p) = \cup\{LV_{entry}(q)|\ q\ follows\ p\ in\ the\ CFG\}$
- $LV_{entry}(p) = gen_{LV}(p) \cup (LV_{exit}(p) \setminus kill_{LV}(p))$

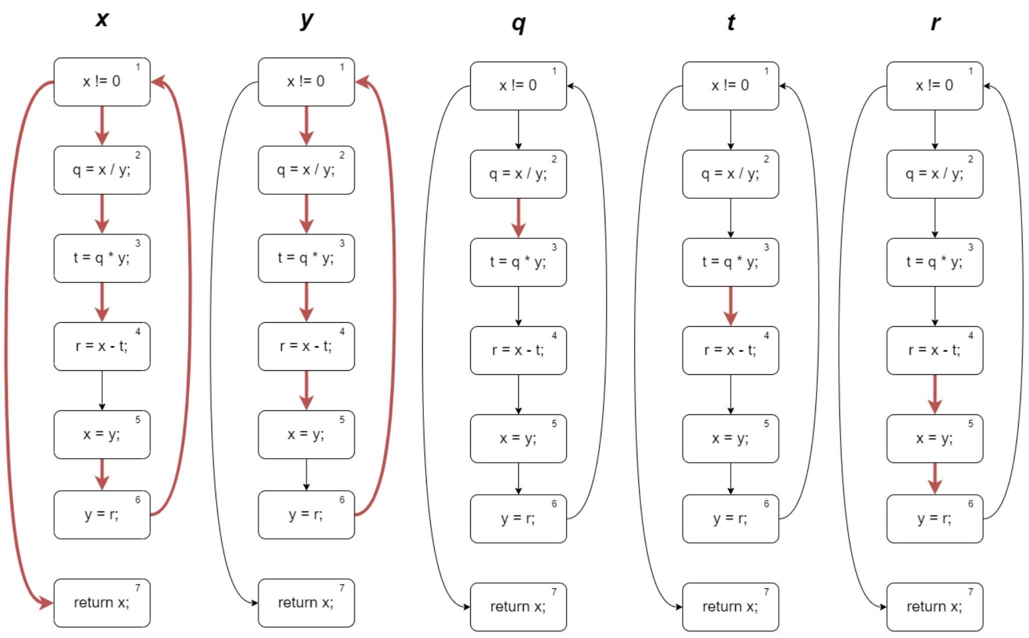
Ordering the nodes, starting from the bottom would be more efficient. We prefer to proceed in this way, because it can be shown better how the algorithms works (update of in set and out set are done almost in two separate iteration)



			1		2		3		4		5	
	use	def	in	out	in	out	in	out	in	out	in	out
1	x		x		x	x y	x y	x y	x y	x y	x y	x y
2	x y	q	x y		x y	q y	x y	q y	x y	q y x	x y	q y x
3	q y	t	q y		q y	x y	q y x	x t	q y x	x t y	q y x	x t y
4	x t	r	x t		x t	y	x t y	y	x t y	y r	x t y	y r
5	y	x	y		y	r	y r	r	y r	r x	y r	r x
6	r	y	r	x	r	x	r x	x y	r x	x y	r x	x y
7	x		x		x		x		x		x	

From the picture we can understand that:

- ***x*** live in every block, and it is redefined in BB 5
- ***y*** live in every block from 1 to 6, and it is redefined in BB 6
- ***q*** live from BB 2 to BB 3
- ***t*** live from BB 3 to BB 4
- ***r*** live from BB 4 to BB 6

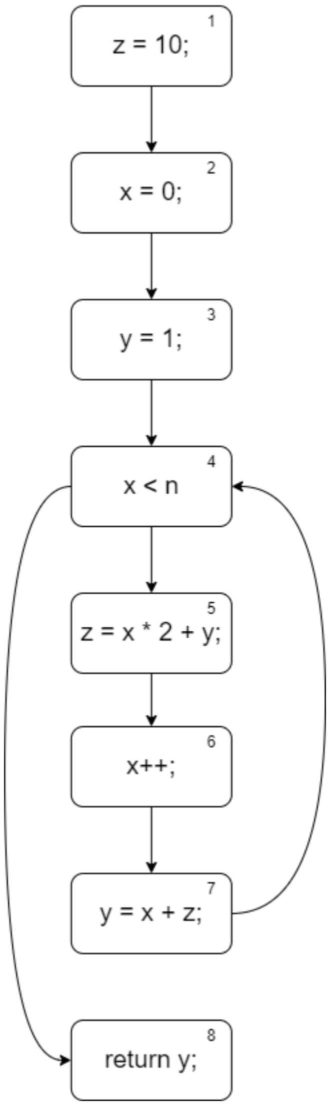


4 Exercise 4: liveness

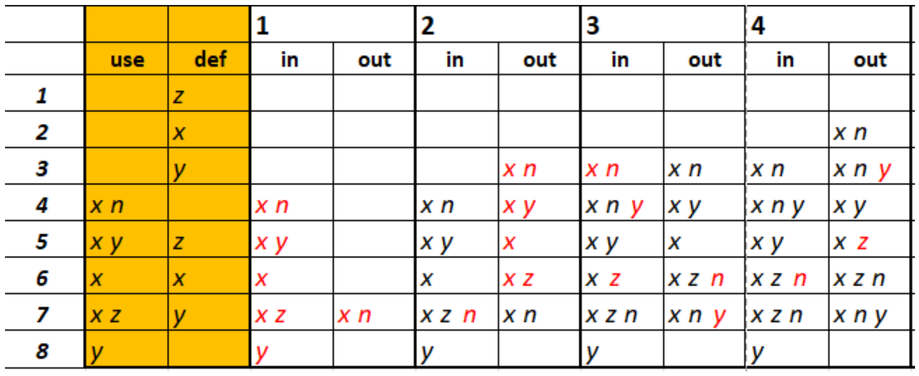
```
z = 10;
x = 0;
y = 1;
while(x < n){
    z = x * 2 + y;
    x++;
    y = x + z;
}
return y;
```

SOLUTION

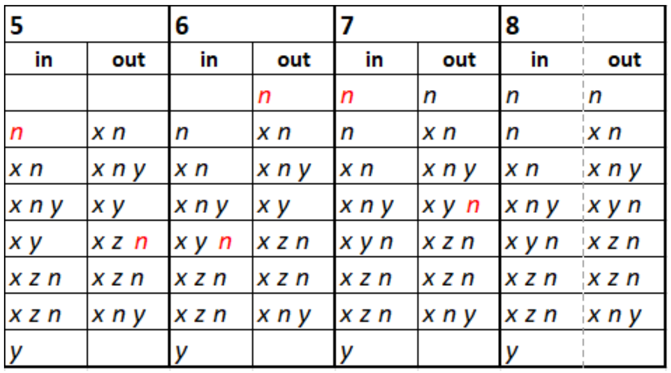
1. Build the control flow graph:



2. Compute the liveness analysis of the program



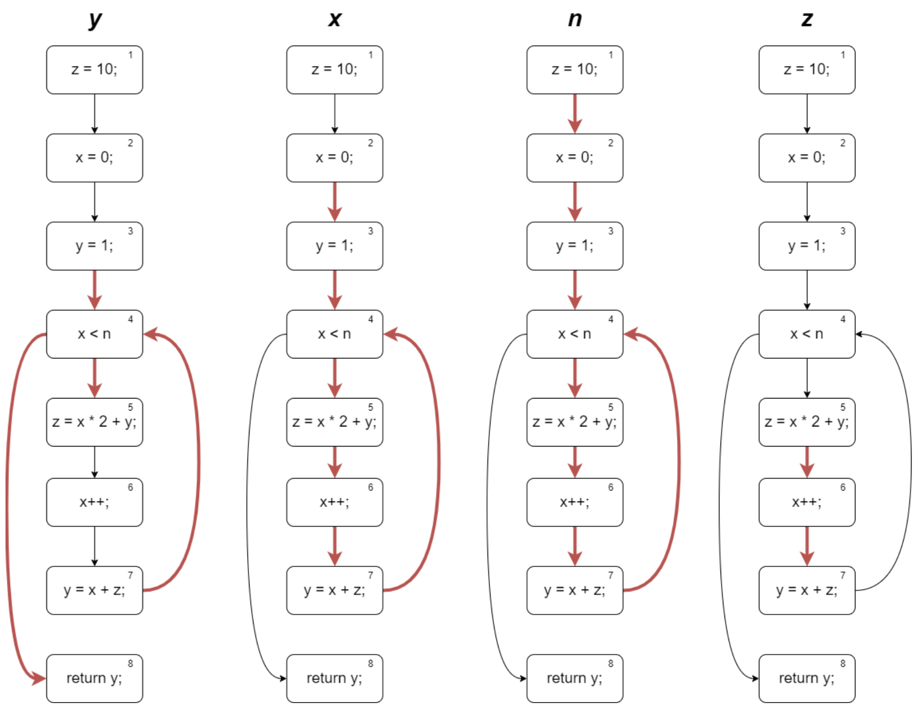
			1		2		3		4	
	use	def	in	out	in	out	in	out	in	out
1		z								
2		x								x n
3		y				x n	x n	x n	x n	x n y
4	x n		x n		x n	x y	x n y	x y	x n y	x y
5	x y	z	x y		x y	x	x y	x	x y	x z
6	x	x	x		x	x z	x z	x z n	x z n	x z n
7	x z	y	x z	x n	x z n	x n	x z n	x n y	x z n	x n y
8	y		y		y		y		y	



5		6		7		8	
in	out	in	out	in	out	in	out
			n	n	n	n	n
n	x n	n	x n	n	x n	n	x n
x n	x n y	x n	x n y	x n	x n y	x n	x n y
x n y	x y	x n y	x y	x n y	x y n	x n y	x y n
x y	x z n	x y n	x z n	x y n	x z n	x y n	x z n
x z n	x z n	x z n	x z n	x z n	x z n	x z n	x z n
x z n	x n y	x z n	x n y	x z n	x n y	x z n	x n y
y		y		y		y	

From the picture we can understand that:

- ***x*** live from BB 2 to BB 7 (include from 7 to 4)
- ***y*** see next picture
- ***n*** live from BB 1 to BB 7 (include from 7 to 4)
- ***z*** live from BB 5 to BB 7



3. Are there variables that can make use of the same register?

From the previous discussion, it is easy to understand that z and y can use the same register to stored the data.