Exercise session 3

3-address Code Generation, Liveness

3.1 Code generation with temporaries

The code we generate will have an unbounded number of temporaries (`tmp0`, `tmp1`, ...) but actual LEIA instructions (`add`, `and`, `let`, ...) except for `snif`

The instruction set and documentation for the LEIA machine can be found in Appendix **??**.

The code generation functions (see Appendix **??**) have the following signatures:

`GenCodeExpr` : AExp $\rightarrow$ Code$^* \times \mathbb{N}$
`GenCodeSmt` : Inst $\rightarrow$ Code*

where $Code*$ is a sequence of 3-address instructions (LEIA with temporaries). As a side effect, the code generation for statements might update a map $Var \rightarrow \mathbb{N}$ (program variable to a temporary where to find its current value).

Auxiliary functions:

`newTemp()` : $\rightarrow \mathbb{N}$
`newLabel()` : $\rightarrow \mathbb{N}$

EXERCISE #1 ▶ **By hand!**
Using the code generation rules for the LEIA machine, generate the three-address LEIA code for the following (mini-while) program:

```
var x1,x2: int;
x1 := 13;
x2 := 7 + x1;
while (x2>x1) do
   x2 := x2-1;
   log (x2);
done
```

EXERCISE #2 ▶ **A new operator for expressions**
Write a code generation rule for the `xor` boolean operator.

EXERCISE #3 ▶ **A new langage construction**
Write a code generation rule for the `repeat S until e` statement.

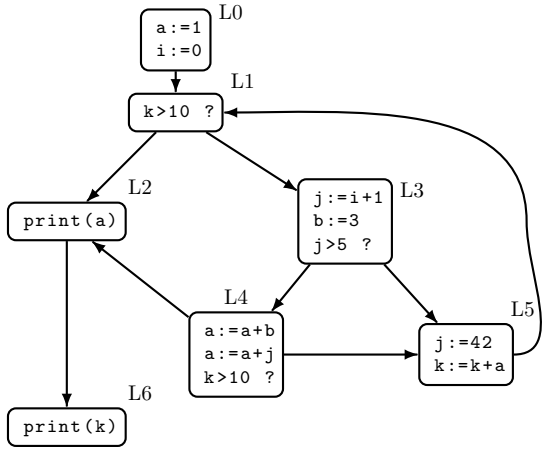
3.2 Liveness analysis

Liveness by hand

EXERCISE #4 ▶ **Liveness by hand - CC 2016**

In Figure 3.1, we give a CFG and we recall that a *variable is alive after a block if there exists a path from this block to one use of this variable that do not contain a definition of it.*

1. (by hand) Fill the array with "out"-alive variables for each block.
2. Remove dead code.



bloc	variables vivantes en sortie du bloc
L0	
L1	
L2	
L3	
L4	
L5	
L6	$\emptyset$

Figure 3.1: CFG and alive variables to complete

Liveness with fixpoint!

Let us recall the notations here: A variable at the left-hand side of an assignement is *killed* by the block. A variable that appears in this bloc is *generated*.

$$LV_{exit}(\ell) = \begin{cases} \emptyset & \text{if } \ell = \text{final} \\ \bigcup\{LV_{entry}(\ell') | (\ell, \ell') \in flow(G)\} & \end{cases}$$

$$LV_{entry}(\ell) = \left(LV_{exit}(\ell) \backslash kill_{LV}(\ell)\right) \cup gen_{LV}(\ell)$$

The sets are initialised to $\emptyset$ and computed iteratively, until reaching a fixpoint.

EXERCISE #5 ▶ **Live variables**
Generate the CFG for the following program:

```
while d>0 then {
    a:=b+c;
    d:=d-b;
    e:=a+f;
    if e>0 then {
        f:=a+d;
        b:=d+f;
    }
    else{
        e:=a-c;
        }
    b:=a+c;
}
```

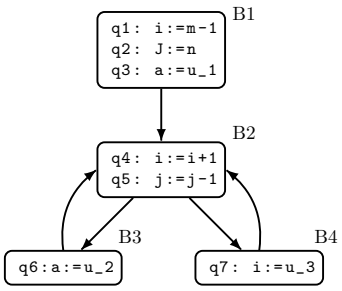
On this CFG:
- Compute Gen, $Kill$ for each block ℓ
- Compute $In(\ell) = LV_{entry}(\ell)$ and $Out(\ell) = LV_{exit}(\ell)$ iteratively.
- Suppress the dead code.

			Step		Step		Step		Step	
ℓ	$kill(\ell)$	$gen(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$

			Step		Step		Step		Step	
ℓ	$kill(\ell)$	$gen(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$

EXERCISE #6 ► **Live Variables**

After code generation, we obtain the following graph:



On this graph, perform liveness analysis and suppress the dead code.

			Step		Step		Step		Step	
ℓ	$kill(\ell)$	$gen(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$

			Step		Step		Step		Step	
ℓ	$kill(\ell)$	$gen(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$