Goal of HW1 is to give you some examples of orthogonalization of concerns. General Comments: there is not right/wrong answer to a question, I would rather consider your homework as solutions to design problems. I could argue that your solution is inefficient and you could argue the same about mine. Grade is established mostly on the rationale you follow to answer questions. Hence it is in your interest to justify all your statements. You can use any kind of sources as long as you reference them all.

Problem description: This is a simple producer-consumer example. Producer and consumer are abstraction of computational blocks that might be very complex: think of two mobile phones for instance. In our case the producer is very simple: it generates random data and we assume that the existence of a library function randomdata() which returns a random object (where the returned type could be whatever you need it to be). Producer generates data on demand. The consumer is just a sink. It receives data and has a way of signaling that the reception is completed.

Assume also that there is a way of checking whether the received data is correct or not, namely there exists a library function checkdata(data) which returns true if the data is correct and false if it is corrupted.

Consider two protocols:

- Simple dropping: consumer checks if the received data is corrupted in which case the data is dropped and a completion signal is send back to the producer.
- Retransmission: consumer checks if the received data is corrupted in which case it asks the producer to send the data again until the reception is successful and a completion signal is sent back to the producer.

QUESTION 1: write the two concurrent processes producer and consumer in the case of simple dropping protocol. You can use finite state machines, process networks, vhd1, verilog, C++, Java, plain English or any other language (that I know about).
QUESTION 2: write the two concurrent processes producer and consumer in the case of retransmission protocol using the same language in question 1.

As you may have noticed, in order to answer question 2 you had to rewrite both processes almost from scratch. This is not a big deal in a simple case like producer-consumer but it breaks reusability. In this case all the producer does is to generate random data. In real systems it might represent a more complex computation and you may want to be able to reuse that component in other projects without touching it. We didn’t separate communication from computation. Producer functionality is to produce data on demand while consumer functionality is to receive data and basically ask for another one:

QUESTION 3: write a process producer that has an input named gendata and an output named data. Producer has to generate a new random data whenever the signal gendata is enabled.

QUESTION 4: write a process consumer that has an input named data and an output named completed. Consumer has to enable the completed signal each time that a data is received.

Note that those two processes can be connected together. The composed system is an autonomous system where producer keeps on producing data and receiver keeps on receiving data. We just modeled the functionality of both processes and connected them together. Now think of the protocol between the two components as two new blocks like in figure .

QUESTION 5: write the up and down converter in the case of simple dropping and retransmission. You can define the protocol handshaking as you like.

Now let’s move to the separation between function and architecture. Assume we have the following directed graph represented with its adjacency list:
QUESTION 6: The adjacency list is of course an implementation (is like memory mapping of program variables). What is the function?

The following algorithm is the classical breadth first search (BFS). It is implemented using the adjacency list representation of a graph. Also it assumes that there is a queue where objects can be stored and retrieved.

**Algorithm 1 BFS**

1: $\text{BFS}(G,s)$
2: for all $u \in V(G)$ do
3:   $\text{color}[u] \leftarrow \text{WHITE}$
4:   $\pi[u] \leftarrow \text{NIL}$
5: end for
6: $\text{color}[s] \leftarrow \text{GRAY}$
7: $\pi[s] \leftarrow \text{NIL}$
8: $Q \leftarrow \{s\}$
9: while $Q \neq \emptyset$ do
10:   $u \leftarrow \text{head}(Q)$
11:   for all $v \in \text{Adj}[u]$ do
12:     if $\text{color}[v] = \text{WHITE}$ then
13:       $\text{color}[v] \leftarrow \text{GRAY}$
14:       $\pi[v] \leftarrow u$
15:       $\text{ENQUEUE}(Q, v)$
16:     end if
17:   end for
18: $\text{DEQUEUE}(Q)$
19: $\text{color}[u] \leftarrow \text{BLACK}$
20: end while

QUESTION 7: write the functional specification of BFS.

QUESTION 8: Based of the functional specification of question 8 and
on the implementation of BFS using adjacency list representation, define a set of library functions that you need to implement BFS. Can you identify the application space, implementation space and platform API?