Problem 1

1. Let $P_1$ be the following reactive module:

   state:
   \[ \text{loc} = \{\text{noacc}, \text{req}, \text{acc}\} \]
   \[ \text{flag}_X = \{\text{down}, \text{up}\} \]

   external:
   \[ \text{flag}_Y = \{\text{down}, \text{up}\} \]

   initial:
   \[ \text{loc} = \text{noacc}; \text{flag}_X = \text{down} \]

   update:
   \[ \text{loc} = \text{noacc} \land \text{flag}_Y = \text{down} \rightarrow \text{loc} = \text{req} \]
   \[ \text{loc} = \text{req} \rightarrow \text{flag}_X = \text{up}; \text{loc} = \text{acc} \]
   \[ \text{loc} = \text{acc} \rightarrow \text{flag}_X = \text{down}; \text{loc} = \text{noacc} \]

   Consider the system $P_1 \parallel P_1[\text{flag}_X/\text{flag}_Y, \text{flag}_Y/\text{flag}_X]$. Does this system ensure the safety property that the two processes are never at the acc location at the same time?

2. Let $P_2$ be the following reactive module:

   state:
   \[ \text{loc} = \{\text{noacc}, \text{req}, \text{acc}\} \]
   \[ \text{flag}_X = \{\text{down}, \text{up}\} \]

   external:
   \[ \text{flag}_Y = \{\text{down}, \text{up}\} \]

   initial:
   \[ \text{loc} = \text{noacc}; \text{flag}_X = \text{down} \]

   update:
   \[ \text{loc} = \text{noacc} \rightarrow \text{flag}_X = \text{up}; \text{loc} = \text{req} \]
   \[ \text{loc} = \text{req} \land \text{flag}_Y = \text{down} \rightarrow \text{loc} = \text{acc} \]
   \[ \text{loc} = \text{acc} \rightarrow \text{flag}_X = \text{down}; \text{loc} = \text{noacc} \]

   Consider the system $P_2 \parallel P_2[\text{flag}_X/\text{flag}_Y, \text{flag}_Y/\text{flag}_X]$. Does this system ensure the safety property “no deadlock” that requires that in every state at least one of the processes can make progress?

3. Design a system $P_3$ that corrects the problems of $P_1$ and $P_2$.

   Does your system ensure the “bounded waiting” property that requires that if a process is waiting for access (i.e. is in the location req), the other process can get access at most a bounded number of times?
Problem 2

Write a functional program \( \exp(m, n) \) which takes two integers \( m \) and \( n \) as inputs, and returns \( m^n \).

Problem 3

\[ 
\textbf{While Semantics} \\

(While Finish) \quad [i]_{st} = 0 \quad \text{while } (i) \text{ do } c \text{ od } st \\
\hline
(While) \quad [i]_{st} \neq 0 \quad st \xrightarrow{c} st' \quad st' \xrightarrow{\text{while } (i) \text{ do } c \text{ od}} st'' \\
\hline
\]

Give a formal semantics for do-until. Do until should have the following syntax \( \text{do } c \text{ until } (i) \text{ od} \). The semantics for do-until is such that after every loop iteration the condition is checked and only if it is true the loop will continue.