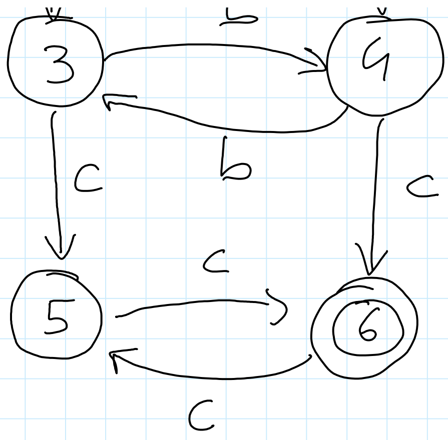
Esercizio 1)

Il linguaggio è regolare.
Un automa che riconosce il linguaggio è il seguente:

$$\begin{tikzcd}
\emptyset \arrow[r, "a"] & 1 \arrow[r, "b"] \arrow[d, bend right, "a"'] & 2 \arrow[r, "c"] \arrow[d, bend right, "b"'] & 3 \arrow[d, bend right, "c"'] \\
 & 4 \arrow[u, bend right, "a"'] \arrow[dr, "b"'] & 5 \arrow[u, bend right, "b"'] \arrow[r, "c"] & 6 \arrow[u, bend right] \\
 & & 7 \arrow[ur, "c"'] \arrow[d, bend right, "b"'] & \\
 & & 8 \arrow[u, bend right, "b"'] \arrow[uuur, bend right, "c"'] &
\end{tikzcd}$$

Oppure il seguente (più compatto):

$$\begin{tikzcd}
\emptyset \arrow[d, "a"] & \\
1 \arrow[r, bend left, "a"] \arrow[d, "b"] & 2 \arrow[l, bend left, "a"] \arrow[d, "b"] \\
3 \arrow[r, "b"] & 4
\end{tikzcd}$$



Una grammatica che genera il linguaggio è la seguente :

$S \to aABC \mid aAbBcC \mid AbBC \mid ABcC$

$A \to a \mid aaA$

$B \to b \mid bbB$

$C \to c \mid ccC$

Esercizio 2)

```
int member (int el, int a[], int dim)
{ int i = ∅;
  int trovato = ∅;
  while (i<dim && !trovato)
    if (a[i] == el) trovato = 1;
      else i++;
  return trovato;
}

int formula (int a[], int dima;
                int b[], int dimb)

{ int i=0;
  int cont = ∅;
  while (i<dima && cont <=2)
  { if (! member (a[i], b, dimb)) cont ++;
     i++;
  }
  return cont <= 2;
}
```

Esercizio 3)

```
let split l n =
    let f x (ult, l1, l2) =
            match ult with
            [] -> ([x], l1, l2)
          | [y] -> if x = n then ([x], y :: l1, l2)
                            else ([x], l1, y :: l2)

    in    match l with
         [] -> ([ ], [ ])

       | x :: xs ->
            let (u, l1, l2) = foldr f ([], [], []) l
            in   (l1, x :: l2) ;;
```

martedì 4 febbraio 2020 12:41

Esercizio 4)

```
let split l n =
  let rec aux l n =
      match l with
      [] -> ([], [])
    | [x] -> ([], [])
    | x :: y :: ys ->
        let (l1, l2) = split (y::ys) n
        in if x = n then (y :: l1, l2)
                    else (l1, y :: l2)

  in match l with
     [] -> ([], [])
   | x :: xs -> let (l1, l2) = aux l n
                in (l1, x :: l2) ;;
```