7 Hoare Logic and Model Checking (rb2018+hk590)

Consider a programming language with commands C consisting of the `skip` no-op command, sequential composition $C_1; C_2$, loops `while` B `do` C for Boolean expressions B, conditionals `if` B `then` C_1 `else` C_2, assignment $X := E$ for program variables X and arithmetic expressions E, heap allocation $X :=$ `alloc`$(E_1, \ldots, E_n)$, heap assignment $[E_1] := E_2$, heap dereference $X := [E]$, and heap location disposal `dispose`(E). Assume `null` $= 0$, and predicates for lists and partial lists:

$$\text{list}(t, []) = (t = \text{null}) \wedge emp$$
$$\text{list}(t, h :: \alpha) = \exists y.(t \mapsto h) * ((t+1) \mapsto y) * \text{list}(y, \alpha)$$
$$\text{plist}(t_1, [], t_2) = (t_1 = t_2) \wedge emp$$
$$\text{plist}(t_1, h :: \alpha, t_2) = \exists y.(t_1 \mapsto h) * ((t_1+1) \mapsto y) * \text{plist}(y, \alpha, t_2)$$

In the following, all triples are linear separation logic triples.

(*a*) Find a command C that satisfies the following separation logic total correctness triple, or explain why no such C exists: $[\top]\ C\ [X \mapsto 1 * X \mapsto 1]$. [2 marks]

(*b*) What property asserted by the $*$ operator makes separation-logic reasoning compositional? Describe briefly how this simplifies reasoning about aliasing pointers, in a way that is not possible using Hoare logic. [3 marks]

(*c*) Define a separation logic predicate $\text{clist}(t, \alpha)$ for pointers t and mathematical lists α that represents a circular singly-linked list. You may use the provided list predicates in your definition. [3 marks]

For each of the following triples, give a loop invariant that would prove it.

(*d*) This command performs a pairwise swap over a list (i.e. swapping every two adjacent elements) by modifying its values. The list is non-empty with even length. The pw_swap function pairwise swaps a mathematical list, e.g. pw_swap $[1,2,3,4] = [2,1,4,3]$.

$\{\text{list}(X, \alpha) \wedge \alpha \neq [] \wedge \exists n.\ \text{length}\ \alpha = 2n\}$
`P` $:=$ `X`;
`while P` $\neq$ `null do`
  (`N1` $:=$ `P`; `N2` $:=$ `[P + 1]`; `NEXT` $:=$ `[N2 + 1]`;
  `TMP` $:=$ `[N1]`; `[N1]` $:=$ `[N2]`; `[N2]` $:=$ `TMP`; `P` $:=$ `NEXT`)
$\{\text{list}(X, \text{pw\_swap}\ \alpha)\}$ [5 marks]

(*e*) This command performs a pairwise swap over the tail of a list that starts with 0 by rearranging its structure in memory. The tail is non-empty with even length.

$\{\text{list}(X, [0]{+}{+}\alpha) \wedge \alpha \neq [] \wedge \exists n.\ \text{length}\ \alpha = 2n\}$
`L` $:=$ `X`; `Y` $:=$ `[X + 1]`;
`while Y` $\neq$ `null do`
  (`N1` $:=$ `[Y + 1]`; `N2` $:=$ `[N1 + 1]`; `[L + 1]` $:=$ `N1`; `[N1 + 1]` $:=$ `Y`; `L` $:=$ `Y`; `Y` $:=$ `N2`);
`[L + 1]` $:=$ `null`
$\{\text{list}(X, [0] +{+} \text{pw\_swap}\ \alpha)\}$ [7 marks]