4.

(1)

a)

$$(\lambda c.\, c(\lambda a.\, \lambda b.\, b))((\lambda a.\, \lambda b.\, \lambda f.\, f\ a\ b)\ p\ q)$$

$$\beta - red \rightarrow ((\lambda a.\, \lambda b.\, \lambda f.\, f\ a\ b)pq)(\lambda a.\, \lambda b.\, b)$$
$$\beta - red \rightarrow ((\lambda b.\, \lambda f.\, f\ p\ b)q)(\lambda a.\, \lambda b.\, b)$$
$$\beta - red \rightarrow (\lambda f.\, f\ p\ q)(\lambda a.\, \lambda b.\, b)$$
$$\beta - red \rightarrow (\lambda a.\, \lambda b.\, b)\ pq$$
$$\beta - red \rightarrow (\lambda b.\, b)q$$
$$\beta - red \rightarrow q$$

b)

$$(\lambda c.\, c(\lambda a.\, \lambda b.\, b))((\lambda a.\, \lambda b.\, \lambda f.\, f\ a\ b)\ p\ q)$$

$$\beta - red \rightarrow (\lambda c.\, c(\lambda a.\, \lambda b.\, b))((\lambda b.\, \lambda f.\, f\ p\ b)q)$$
$$\beta - red \rightarrow (\lambda c.\, c(\lambda a.\, \lambda b.\, b))(\lambda f.\, f\ p\ q)$$
$$\beta - red \rightarrow \lambda f.\, f\ p\ q(\lambda a.\, \lambda b.\, b)$$
$$\beta - red \rightarrow (\lambda a.\, \lambda b.\, b)p\ q$$
$$\beta - red \rightarrow (\lambda b.\, b)q$$
$$\beta - red \rightarrow q$$

(2)

a)

$$\Big(\lambda x.\, \lambda y.\, \Big(mul\ x\ \big((\lambda x.\, (add\ x\ 3))y\ \big)\Big)\Big)\ 7\ 8$$

$$\alpha - reduction \rightarrow \Big(\lambda x.\, \lambda y.\, \Big(\ mulx\ \big((\lambda t.\, (\ add\ t\ 3))y\ \big)\Big)\Big)\ 7\ 8$$

$$\beta - reduction \rightarrow \Big(\lambda y.\, \Big(mul\ 7\ \big((\lambda t.\, (add\ t\ 3))y\ \big)\Big)\Big)\ 8$$

$$\beta - reduction \rightarrow \big(mul\ 7\big((\lambda t.\, (add\ t\ 3)8)\big)\big)$$

$$\beta - reduction \rightarrow (\ mul\ 7\ \big(\big(\ add\ (\ 8\ 3)\big)\big))$$

$$\delta - red \rightarrow 7 * (\ 8 + 3) =\ 77$$

b)

$$\Big(\lambda x.\, \lambda y.\, \Big(mul\ x\ \big((\lambda x.\, (add\ x\ 3))y\ \big)\Big)\Big)\ 7\ 8$$

$$\alpha - reduction \rightarrow \big(\lambda x.\, \lambda y(\ mul\ x\ \big((\lambda t.\, (add\ t\ 3)\ y)\big)\big)\ 7\ 8$$

$$\beta - reduction \rightarrow \Big(\lambda x.\, \lambda y\ \Big(\ mul\ x\ \big(add\ (\ y\ 3)\big)\Big)\Big)\ 7\ 8$$

$$\beta - reduction \rightarrow\ \lambda y.\, (\ mul\ 7\ (add\ (\ y\ 3))))8$$

$$\beta - reduction \rightarrow (mul\ 7\ \big(\ add\ (\ 8\ 3\ )\big))$$

$\delta - reduction \rightarrow 77$

(3)

a)

printf(k) =>

$k = \lambda x.(add\ x\ 1)\ 1$

= 2 (by beta reduction)

printf(m) =>

$m = (\lambda g.g\ 5)(\lambda x.(x + 3))$

$m = \lambda x.(x + 3)5$

= 8

printf(k) =>

swap(k,m) -> k = 8

printf(k,m) =>

5,11.

The pi function is meant to, swap the number and increment by 1 each. Since it's done 3 times.

b)

I agree. In this program, both equations : f1 and f2 are still valid and giving the same answer with any reduction, normal order or applicative order. Therefore, I think it's still fine if we change it into k = (f1 m) .

c)

Scope of variable t is only within the braces of {} after pi (x,y).

Scope of variable x is only within each f, such as f1 and f2.

Lifetime of variable t and x are both automati as the allocation start when it enters statement block and ends when exiting the statement block.

d)

Since the programming language paradigm follows lambda calculus, which is a functional programming style, I would prefer to see using dynamic programming , where when you call a function, the value of a variable depends on the calling function's variable value. Since in functional programming language, recursion is often used to do pattern matching, it would be much easier to use dynamic programming rather than having to pass parameter all the time for the same kind of purpose.

And at the same time, just like I state above about the lifetime and scope of variable of t and x, using dynamic scoping would be much easier because every block will immediately set the value and so using dynamic programming, you will use the corresponding value of t and x in each sub-function/routine call.