1 The `ListRemove(x,k)` function, which removes all occurrences of the element `k` from the singly-linked list at `x`, is implemented as follows:

```
ListRemove(x,k) {
    if (x = null) {
      skip
    } else {
      t := [x];
      n := [x + 1];
      if (t = k) {
        dispose(x);
        dispose(x + 1);
        x := ListRemove(n,k)
      } else {
        t := ListRemove(n,k);
        [x + 1] := t
      };
    };
    return x
  }
```

Consider the singly-linked list predicate $\mathsf{list}(E,\alpha)$ defined in the lectures, and the additional predicates:

$$\begin{aligned}\mathsf{lmem}(E,\alpha,b) \triangleq\ & (E \mathrel{\dot\in} \mathsf{Val} * \alpha \doteq \epsilon * b \doteq \mathtt{false}) \lor \\ & (\exists \alpha'.\ \alpha \doteq E : \alpha' * b \doteq \mathtt{true}) \lor \\ & (\exists a, \alpha'.\ \alpha \doteq a : \alpha' * a \mathrel{\dot{\neq}} E * \mathsf{lmem}(E,\alpha',b))\end{aligned}$$

and

$$\begin{aligned}\mathsf{lsub}(\beta,\alpha) \triangleq\ & (\beta \doteq \epsilon * \alpha \mathrel{\dot\in} \mathsf{List}) \lor \\ & (\exists b, \beta', \alpha'.\ \beta \doteq b : \beta' * \alpha \doteq b : \alpha' * \mathsf{lsub}(\beta',\alpha')) \lor \\ & (\exists b, \beta', a, \alpha'.\ \beta \doteq b : \beta' * \alpha \doteq a : \alpha' * b \mathrel{\dot{\neq}} a * \mathsf{lsub}(\beta,\alpha'))\end{aligned}$$

and the following five valid entailments:

(E1) $\vdash \mathsf{lmem}(E,\alpha,b) * \alpha \doteq \epsilon \Rightarrow \mathsf{lmem}(E,\alpha,\mathtt{false})$

(E2) $\vdash E \mathrel{\dot\in} \mathsf{Val} * \alpha \mathrel{\dot\in} \mathsf{List} \Rightarrow \exists b.\ \mathsf{lmem}(E,\alpha,b)$

(E3) $\vdash \alpha \mathrel{\dot\in} \mathsf{List} \Rightarrow \mathsf{lsub}(\alpha,\alpha)$

(E4) $\vdash E \mathrel{\dot\in} \mathsf{Val} * \mathsf{lsub}(\beta,\alpha) \Rightarrow \mathsf{lsub}(E : \beta, E : \alpha)$

(E5) $\vdash E \mathrel{\dot\in} \mathsf{Val} * \mathsf{lsub}(\beta,\alpha) \Rightarrow \mathsf{lsub}(\beta, E : \alpha)$

a Explain in words the meaning of the predicates $\mathsf{lmem}(E,\alpha,b)$ and $\mathsf{lsub}(\beta,\alpha)$.

The list membership predicate, $\mathsf{lmem}(E,\alpha,b)$ uses the Boolean b to capture whether a given logical expression E is a member of the given mathematical list α. (1 mark)

The sublist predicate, $\mathsf{lsub}(\beta,\alpha)$, states that all of the elements of the mathematical list β appear in the same order, but not necessarily consecutively, in the mathematical list α. (1 mark)

Marks: **2**

b Highlighting any of the entailments E1–E5 used, give a proof derivation of the judgement:

$$\begin{array}{l}\Gamma \vdash \{\ \mathtt{x} \mapsto k, \mathtt{n} * \mathsf{list}(\mathtt{n},\alpha') * \alpha \doteq k : \alpha' * \mathtt{k} \doteq k\ \}\\ \qquad \mathtt{dispose(x);\ dispose(x+1);\ x := ListRemove(n,k)}\\ \qquad \{\ \exists\beta.\ \mathsf{list}(\mathtt{x},\beta) * \mathsf{lsub}(\beta,\alpha) * \mathsf{lmem}(k,\beta,\mathtt{false})\ \}\end{array}$$

where

$$\begin{array}{l}\Gamma = \{\ \ \{\ P : \mathsf{list}(\mathtt{x},\alpha) * \mathtt{k} \doteq k\ \}\\ \qquad\quad \mathtt{ListRemove(x,k)}\\ \qquad\quad \{\ Q : \exists\beta.\ \mathsf{list}(\mathtt{ret},\beta) * \mathsf{lsub}(\beta,\alpha) * \mathsf{lmem}(k,\beta,\mathtt{false})\ \}\}\end{array}$$

Let

$$\begin{array}{rcl} C_1 & \triangleq & \mathtt{dispose(x)}\\ C_2 & \triangleq & \mathtt{dispose(x+1)}\\ C_3 & \triangleq & \mathtt{x := ListRemove(n,k)}\\ P_1 & \triangleq & \mathsf{list}(\mathtt{n},\alpha') * \alpha \doteq k : \alpha' * \mathtt{k} \doteq k\\ Q_1 & \triangleq & \exists\beta.\ \mathsf{list}(\mathtt{x},\beta) * \mathsf{lsub}(\beta,\alpha) * \mathsf{lmem}(k,\beta,\mathtt{false})\end{array}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \{\mathtt{x}\mapsto k\}\ C_1\ \{\mathsf{emp}\}}\ \textit{dispose}}{\Gamma \vdash \{\mathtt{x}\mapsto k * \mathtt{x}+1 \mapsto \mathtt{n} * P_1\}\ C_1\ \{\mathsf{emp} * \mathtt{x}+1\mapsto \mathtt{n} * P_1\}}\ \textit{frame}}{\Gamma \vdash \{\mathtt{x}\mapsto k,\mathtt{n} * P_1\}\ C_1\ \{\mathtt{x}+1\mapsto\mathtt{n} * P_1\}}\ \textit{cons} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma \vdash \{\mathtt{x}+1\mapsto \mathtt{n}\}\ C_2\ \{\mathsf{emp}\}}\ \textit{dispose}}{\Gamma\vdash\{\mathtt{x}+1\mapsto\mathtt{n} * P_1\}\ C_2\ \{\mathsf{emp} * P_1\}}\ \textit{frame} \quad (\dagger)}{\Gamma\vdash\{\mathtt{x}+1\mapsto\mathtt{n} * P_1\}\ C_2;C_3\ \{Q_1\}}\ \textit{seq}}{\Gamma\vdash\{\mathtt{x}\mapsto k,\mathtt{n} * P_1\}\ C_1;C_2;C_3\ \{Q_1\}}\ \textit{seq}$$

where $(\dagger)$ *is as follows:*

$$\dfrac{\dfrac{\dfrac{\{P\}\ \mathtt{ListRemove(x,k)}\ \{Q\} \in \Gamma}{\Gamma\vdash\{\mathtt{n} \mathrel{\dot\in} \mathsf{Val} * \mathtt{k} \mathrel{\dot\in} \mathsf{Val} * \mathsf{list}(\mathtt{n},\alpha') * \mathtt{k} \doteq k\}\ C_3\ \{\exists\beta.\ \mathsf{list}(\mathtt{x},\beta) * \mathsf{lsub}(\beta,\alpha') * \mathsf{lmem}(k,\beta,\mathtt{false})\}}\ \textit{fcall}}{\Gamma\vdash\{\mathtt{n} \mathrel{\dot\in} \mathsf{Val} * \mathtt{k} \mathrel{\dot\in} \mathsf{Val} * \mathsf{list}(\mathtt{n},\alpha') * \mathtt{k}\doteq k * \alpha \doteq k : \alpha'\}\ C_3\ \{(\exists\beta.\ \mathsf{list}(\mathtt{x},\beta) * \mathsf{lsub}(\beta,\alpha') * \mathsf{lmem}(k,\beta,\mathtt{false})) * \alpha \doteq k : \alpha'\}}\ \textit{frame} \quad (E5)}{\Gamma\vdash\{\mathsf{emp} * P_1\}\ C_3\ \{Q_1\}}\ \textit{cons}$$

Sequencing carries 1 mark; the derivation tree of each command carries 1 mark for 3 marks in total; noting the use of entailment (E5) carries 1 mark.

Marks: **5**

c Using Γ and the result from part (1b), and highlighting any of the entailments E1–E5 used, give a proof sketch that the `ListRemove(x,k)` function satisfies its specification:

$$\{P\}\ \mathtt{ListRemove(x,k)}\ \{Q\}$$

```
$\{\ P : \mathsf{list}(\mathtt{x},\alpha) * \mathtt{k} \doteq k\ \}$
ListRemove(x,k) {
  $\{\ \mathsf{list}(\mathtt{x},\alpha) * \mathtt{k} \doteq k * \mathtt{t},\mathtt{n} \doteq \mathtt{null}\ \}$
  if (x = null) {
    $\{\ \mathsf{list}(\mathtt{x},\alpha) * \mathtt{k} \doteq k * \mathtt{t},\mathtt{n} \doteq \mathtt{null} * \mathtt{x} \doteq \mathtt{null}\ \}$
    $\{\ \mathsf{list}(\mathtt{x},\alpha) * k \in \mathsf{Val} * \alpha \doteq \epsilon\ \}$
    $\{\ \exists m.\ \mathsf{list}(\mathtt{x},\alpha) * \mathsf{lmem}(k,\alpha,m) * \alpha \doteq \epsilon\ \}$   // By E2
    $\{\ \mathsf{list}(\mathtt{x},\alpha) * \mathsf{lmem}(k,\alpha,\mathtt{false})\ \}$   // By E1
    $\{\ \mathsf{list}(\mathtt{x},\alpha) * \mathsf{lsub}(\alpha,\alpha) * \mathsf{lmem}(k,\alpha,\mathtt{false})\ \}$   // By E3
    // Consequence: introduce $\beta$ to match post-condition, keep $\alpha$ only as the second parameter of lsub
    $\{\ Q[\mathtt{x}/\mathtt{ret}]\ \}$
    skip
    $\{\ Q[\mathtt{x}/\mathtt{ret}]\ \}$
  } else {
    $\{\ \mathsf{list}(\mathtt{x},\alpha) * \mathtt{k} \doteq k * \mathtt{t},\mathtt{n} \doteq \mathtt{null} * \mathtt{x} \not\doteq \mathtt{null}\ \}$
    $\{\ \exists a,y,\alpha'.\ \mathtt{x} \mapsto a,y * \mathsf{list}(y,\alpha') * \alpha \doteq a:\alpha' * \mathtt{k} \doteq k * \mathtt{t},\mathtt{n} \doteq \mathtt{null}\ \}$
    t := [x];
    $\{\ \exists a,y,\alpha'.\ \mathtt{x} \mapsto a,y * \mathsf{list}(y,\alpha') * \alpha \doteq a:\alpha' * \mathtt{k} \doteq k * \mathtt{t} \doteq a * \mathtt{n} \doteq \mathtt{null}\ \}$
    n := [x + 1];
    $\{\ \exists a,\alpha'.\ \mathtt{x} \mapsto a,\mathtt{n} * \mathsf{list}(\mathtt{n},\alpha') * \alpha \doteq a:\alpha' * \mathtt{k} \doteq k * \mathtt{t} \doteq a\ \}$
    if (t = k) {
      $\{\ \exists \alpha'.\ \mathtt{x} \mapsto k,\mathtt{n} * \mathsf{list}(\mathtt{n},\alpha') * \alpha \doteq k:\alpha' * \mathtt{k} \doteq k * \mathtt{t} \doteq k\ \}$
      $\{\ \exists \alpha'.\ \mathtt{x} \mapsto k,\mathtt{n} * \mathsf{list}(\mathtt{n},\alpha') * \alpha \doteq k:\alpha' * \mathtt{k} \doteq k\ \}$
      dispose(x); dispose(x + 1); x := ListRemove(n,k)
      $\{\ Q[\mathtt{x}/\mathtt{ret}]\ \}$
    } else {
      $\{\ \exists a,\alpha'.\ \mathtt{x} \mapsto a,\mathtt{n} * \mathsf{list}(\mathtt{n},\alpha') * \alpha \doteq a:\alpha' * \mathtt{k} \doteq k * \mathtt{t} \doteq a * a \not\doteq k\ \}$
      $\{\ \exists a,\alpha'.\ \mathtt{x} \mapsto a,\mathtt{n} * \mathsf{list}(\mathtt{n},\alpha') * \alpha \doteq a:\alpha' * \mathtt{k} \doteq k * a \not\doteq k\ \}$
      t := ListRemove(n,k);
      $\{\ \exists a,\alpha',\beta'.\ \mathtt{x} \mapsto a,\mathtt{n} * \mathsf{list}(\mathtt{t},\beta') * \alpha \doteq a:\alpha' * \mathsf{lsub}(\beta',\alpha') * \mathsf{lmem}(k,\beta',\mathtt{false}) * a \not\doteq k\ \}$
      [x + 1] := t
      $\{\ \exists a,\alpha',\beta'.\ \mathtt{x} \mapsto a,\mathtt{t} * \mathsf{list}(\mathtt{t},\beta') * \alpha \doteq a:\alpha' * \mathsf{lsub}(\beta',\alpha') * \mathsf{lmem}(k,\beta',\mathtt{false}) * a \not\doteq k\ \}$
      $\{\ \exists a,\alpha',\beta'.\ \mathsf{list}(\mathtt{x},a:\beta') * \alpha \doteq a:\alpha' * \mathsf{lsub}(\beta',\alpha') * \mathsf{lmem}(k,\beta',\mathtt{false}) * a \not\doteq k\ \}$
      $\{\ \exists a,\alpha',\beta'.\ \mathsf{list}(\mathtt{x},a:\beta') * \mathsf{lsub}(a:\beta',\alpha) * \mathsf{lmem}(k,\beta',\mathtt{false}) * a \not\doteq k\ \}$   // By E4
      $\{\ \exists a,\beta'.\ \mathsf{list}(\mathtt{x},a:\beta') * \mathsf{lsub}(a:\beta',\alpha) * \mathsf{lmem}(k,a:\beta',\mathtt{false})\ \}$
      $\{\ Q[\mathtt{x}/\mathtt{ret}]\ \}$
    };
    $\{\ Q[\mathtt{x}/\mathtt{ret}]\ \}$
  };
  $\{\ Q[\mathtt{x}/\mathtt{ret}]\ \}$
  return x
}
$\{\ Q : \exists \beta.\ \mathsf{list}(\mathsf{ret},\beta) * \mathsf{lsub}(\beta,\alpha) * \mathsf{lmem}(k,\beta,\mathtt{false})\ \}$
```

Mark breakdown: 1 mark for entering and exiting the function; 4 marks for each use of the entailments; 4 marks for rest of proof sketch.

Marks: **9**

d Consider the specification of the `ListRemove(x,k)` function given in part 1c.

i) Explain in words how the post-condition of the given specification loses information about the relationship between β and α.

ii) Explain in words, possibly by introducing additional predicates (mathematically or in words), how the post-condition of the given specification can be strengthened to precisely describe the relationship between β and α.

i) From the post-condition, we know that the resulting list β is a sublist of α and that it does not contain the removed element k. What is lost, however, is the fact that no element other than k has been removed. (1 mark)

ii) It would be sufficient to have a predicate that counts the occurrences of an element in a list: for example

$$\begin{aligned}\mathsf{lcount}(E,\alpha,n) \triangleq\ & (E \mathrel{\dot\in} \mathsf{Val} * \alpha \doteq \epsilon * n \doteq 0) \lor \\ & (\exists \alpha', n'.\, \alpha \doteq E : \alpha' * \mathsf{lcount}(E,\alpha',n') * n \doteq n'+1) \lor \\ & (\exists a, \alpha'.\, \alpha \doteq a : \alpha' * a \mathrel{\dot{\neq}} E * \mathsf{lcount}(E,\alpha',n))\end{aligned}$$

and have the post-condition be

$$\exists \beta, n.\, \mathsf{list}(\mathsf{ret},\beta) * \mathsf{lsub}(\beta,\alpha) * \mathsf{lmem}(k,\beta,\texttt{false}) * \mathsf{lcount}(k,\alpha,n) * |\beta| \doteq |\alpha| - n$$

From there, we can entail that the all elements of α that are different from k appear the same number of times in α and in β, which, together with $\mathsf{lsub}(\beta,\alpha)$, fully characterises their relationship.

(3 marks)

Marks: **4**

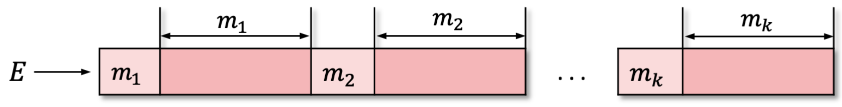
The four parts carry, respectively, 10%, 25%, 45%, and 20% of the marks.

2 Consider the buffer predicate, $\mathsf{buf}(E, m)$, and the buffer list predicate, $\mathsf{bufList}(E, \alpha)$, defined by:

$$\mathsf{buf}(E, m) \triangleq E \mapsto m * \circledast_{1 \le i \le m}(E + i \mapsto -)$$

$$\begin{aligned}\mathsf{bufList}(E, \alpha) \triangleq\ & (E \mathrel{\dot\in} \mathbb{N} * \alpha \doteq \epsilon) \vee \\ & (\exists m, \alpha'.\ \alpha \doteq m : \alpha' * \mathsf{buf}(E, m) * \\ & \qquad \mathsf{bufList}(E + m + 1, \alpha'))\end{aligned}$$

The buffer predicate, $\mathsf{buf}(E, m)$, declares a buffer of variable length, with the value at address E giving the buffer length, and the cells from $E + 1$ to $E + m$ providing the actual contents held by the buffer, which are not described by the predicate. The buffer list predicate, $\mathsf{bufList}(E, \alpha)$, describes a list of buffers starting from address E, where the elements of α describe the individual lengths of the buffers in the list and $|\alpha|$ denotes the number of buffers in the list. Its heap diagram is as follows:



$\alpha = [m_1, m_2, \ldots, m_k]$

a Let $\Sigma(\alpha)$ denote the sum of all elements of the mathematical list of natural numbers, α. Consider the following entailments:

(E1) $\vdash\ \mathsf{buf}(E, m) \Rightarrow E \in \mathbb{N} \wedge m \in \mathbb{N}$
(E2) $\vdash\ \mathsf{bufList}(E, [m]) \Leftrightarrow \mathsf{buf}(E, m)$
(E3) $\vdash\ \mathsf{bufList}(E, \alpha) \Rightarrow \circledast_{0 \le i < \Sigma(\alpha) + |\alpha|}(E + i \mapsto -)$

Assuming (E1), prove that (E2) and (E3) hold.

The entailment (E2) is proven by unfolding $\mathsf{bufList}$ *and using (E1).*

$$\begin{array}{ll}
\mathsf{bufList}(E, [m]) & \\
\Leftrightarrow (\exists m', \alpha'.\ [m] \doteq m' : \alpha' * \mathsf{buf}(E, m') * \mathsf{bufList}(E + m' + 1, \alpha')) & (\textit{unfold}\ \mathsf{bufList}) \\
\Leftrightarrow \mathsf{buf}(E, m) * \mathsf{bufList}(E + m + 1, \epsilon) & (\textit{cons}) \\
\Leftrightarrow \mathsf{buf}(E, m) * (E + m + 1 \mathrel{\dot\in} \mathbb{N} * \epsilon \doteq \epsilon) & (\textit{unfold}\ \mathsf{bufList} + \textit{cons}) \\
\Leftrightarrow \mathsf{buf}(E, m) * E \mathrel{\dot\in} \mathbb{N} * m \mathrel{\dot\in} \mathbb{N} & (\textit{cons}) \\
\Leftrightarrow \mathsf{buf}(E, m) & (\textit{cons, using (E1)})
\end{array}$$

(2 marks)

The entailment (E3) is proven by induction on α.
Base case: $\alpha = \epsilon$.

To show: $\mathsf{bufList}(E, \epsilon) \Rightarrow \circledast_{0 \le i < \Sigma(\epsilon) + |\epsilon|}(E + i \mapsto -)$

$$
\begin{array}{lr}
\mathsf{bufList}(E, \epsilon) & \\
\Rightarrow E \mathrel{\dot{\in}} \mathbb{N} * \epsilon \doteq \epsilon & (\textit{unfold}\ \mathsf{bufList}) \\
\Rightarrow \circledast_{0 \le i < 0}(E + i \mapsto -) & (\textit{cons}) \\
\Rightarrow \circledast_{0 \le i < \Sigma(\epsilon) + |\epsilon|}(E + i \mapsto -) & (\textit{cons})
\end{array}
$$

Inductive case: $\alpha = m : \alpha'$, *for some* m, α'.
To show: $\mathsf{bufList}(E, m : \alpha') \Rightarrow \circledast_{0 \le i < \Sigma(m:\alpha') + |m:\alpha'|}(E + i \mapsto -)$

$$
\begin{array}{lr}
\mathsf{bufList}(E, m : \alpha') & \\
\Rightarrow \mathsf{buf}(E, m) * \mathsf{bufList}(E + m + 1, \alpha') & (\textit{unfold}\ \mathsf{bufList}) \\
\Rightarrow \mathsf{buf}(E, m) * \circledast_{0 \le i < \Sigma(\alpha') + |\alpha'|}(E + m + 1 + i \mapsto -) & (\mathbf{IH}) \\
\Rightarrow E \mapsto m * \circledast_{1 \le i \le m}(E + i \mapsto -) * \circledast_{0 \le i < \Sigma(\alpha') + |\alpha'|}(E + m + 1 + i \mapsto -) & (\textit{unfold}\ \mathsf{buf}) \\
\Rightarrow \circledast_{0 \le i < m + 1 + \Sigma(\alpha') + |\alpha'|}(E + i \mapsto -) & (\textit{cons}) \\
\Rightarrow \circledast_{0 \le i < \Sigma(m:\alpha') + |m:\alpha'|}(E + i \mapsto -) & (\textit{cons})
\end{array}
$$

1 mark for the overall inductive structure, 1 mark for the base case and 2 marks for the inductive case.

Marks: **6**

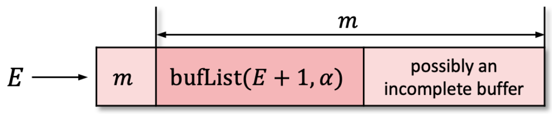
b Give the proof derivation for the judgement:

$$
\begin{array}{l}
\vdash \{\ \mathsf{buf}(\mathtt{x}, m) * \mathsf{c} \doteq -\ \} \\
\quad \mathsf{c} := [\mathtt{x}] \\
\quad \{\ \mathsf{buf}(\mathtt{x}, m) * \mathsf{c} \doteq m\ \}
\end{array}
$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{c} \notin \mathsf{pv}(c)}{\Gamma \vdash \{\mathsf{c} \doteq c * \mathtt{x} \mapsto m\}\ \mathsf{c} := [\mathtt{x}]\ \{\mathsf{c} \doteq m * \mathtt{x} \mapsto m\}}\ \textit{lookup}}{\Gamma \vdash \left\{\mathsf{c} \doteq c * \mathtt{x} \mapsto m * \circledast_{1 \le i \le m}(\mathtt{x} + i \mapsto -)\right\}\ \mathsf{c} := [\mathtt{x}]\ \left\{\mathsf{c} \doteq m * \mathtt{x} \mapsto m * \circledast_{1 \le i \le m}(\mathtt{x} + i \mapsto -)\right\}}\ \textit{frame}}{\Gamma \vdash \left\{\exists c.\ \mathsf{c} \doteq c * \mathtt{x} \mapsto m * \circledast_{1 \le i \le m}(\mathtt{x} + i \mapsto -)\right\}\ \mathsf{c} := [\mathtt{x}]\ \left\{\exists c.\ \mathsf{c} \doteq m * \mathtt{x} \mapsto m * \circledast_{1 \le i \le m}(\mathtt{x} + i \mapsto -)\right\}}\ \textit{exists}}{\vdash \{\mathsf{buf}(\mathtt{x}, m) * \mathsf{c} \doteq -\}\ \mathsf{c} := [\mathtt{x}]\ \{\mathsf{buf}(\mathtt{x}, m) * \mathsf{c} \doteq m\}}\ \textit{cons}
$$

Marks: **2**

c Recall that the buffer predicate, $\mathsf{buf}(E, m)$, declares a cell at address E with value m, and cells from $E + 1$ to $E + m$, whose contents have not been described. Assume a non-empty buffer whose contents can be viewed as a buffer list, possibly with an incomplete buffer at the end:



The function countBuffers(x), given below, counts the number of buffers present in the contents of the buffer at x:

```
countBuffers(x) {
  m := [x]; x := x + 1;
  r := 0; stop := false;
  while ((m > 0) and (stop = false)) {
    c := [x];
    if (m < c + 1) {
      stop := true
    } else {
      r := r + 1;
      x := x + c + 1;
      m := m − c − 1
    }
  };
  return r
}
```

Give a proof sketch that the countBuffers(x) function satisfies the specification:

$$\begin{array}{l}\{\ \mathtt{x} \doteq x * x \mapsto m * \mathsf{bufList}(x+1, \alpha) * m \doteq \Sigma(\alpha) + |\alpha|\ \}\\ \quad \mathtt{countBuffers(x)}\\ \{\ \mathsf{buf}(x, \Sigma(\alpha) + |\alpha|) * \mathtt{ret} \doteq |\alpha|\ \}\end{array}$$

with function context specification ∅, using the while loop invariant

$$\left\{\ P_{inv} : \begin{array}{l} \exists \beta, \gamma.\ x \mapsto m * \mathsf{bufList}(x+1, \beta) * \mathsf{bufList}(\mathtt{x}, \gamma) * \alpha \doteq \beta \cdot \gamma * \\ \quad m \doteq \Sigma(\alpha) + |\alpha| * \mathtt{x} \doteq x + 1 + \Sigma(\beta) + |\beta| * \mathtt{m} \doteq \Sigma(\gamma) + |\gamma| * \\ \quad \mathtt{r} \doteq |\beta| * \mathtt{stop} \doteq \mathtt{false} * \mathtt{c} \doteq - \end{array}\ \right\}$$

and the result from part (2b), and inlining brief descriptions of any non-trivial entailments used and indicating any entailments used from Part 2a. Commands written on the same line may be specified at the same time.

(Note that, to simplify the proof sketch, the pre-condition is restricted to the case in which the buffer contents represent a complete buffer list.)

The proof sketch is as follows.

$\{\ \mathtt{x} \doteq x * x \mapsto m * \mathsf{bufList}(x+1, \alpha) * m \doteq \Sigma(\alpha) + |\alpha|\ \}$

countFields(x) {

  $\{\ \mathtt{x} \doteq x * x \mapsto m * \mathsf{bufList}(x+1, \alpha) * m \doteq \Sigma(\alpha) + |\alpha| * \mathtt{m}, \mathtt{r}, \mathtt{stop}, \mathtt{c} \doteq \mathtt{null}\ \}$

  m := [x]; x := x + 1;

  $\{\ \mathtt{x} \doteq x + 1 * x \mapsto m * \mathsf{bufList}(\mathtt{x}, \alpha) * m \doteq \Sigma(\alpha) + |\alpha| * \mathtt{m} \doteq m * \mathtt{r}, \mathtt{stop}, \mathtt{c} \doteq \mathtt{null}\ \}$

  r := 0; stop := false;

```
{ $x \mapsto m * \mathsf{bufList}(\mathtt{x}, \alpha) * m \doteq \Sigma(\alpha) + |\alpha| * \mathtt{m} \doteq m *$
  $\mathtt{r} \doteq 0 * \mathtt{stop} \doteq \mathtt{false} * \mathtt{x} \doteq x + 1 * \mathtt{c} \doteq \mathtt{null}$ }
// Initial value of $\beta$ is $\epsilon$; initial value of $\gamma$ is $\alpha$; forget actual value of c
{ $P_{inv}$ }
while (($\mathtt{m} > 0$) and ($\mathtt{stop} = \mathtt{false}$)) {
  { $P_{inv} \wedge (\mathtt{m} > 0 \wedge \mathtt{stop} = \mathtt{false})$ }
  // Since $\mathtt{m} > 0$, we have that $\gamma$ is not empty and can unfold $\mathsf{bufList}(x, \gamma)$
  { $\exists \beta, \gamma, m', \gamma'.\ x \mapsto m * \mathsf{bufList}(x+1, \beta) * \gamma \doteq m' : \gamma' * \mathsf{buf}(\mathtt{x}, m') * \mathsf{bufList}(\mathtt{x} + m' + 1, \gamma') *$
            $\alpha \doteq \beta \cdot \gamma * m \doteq \Sigma(\alpha) + |\alpha| * \mathtt{x} \doteq x + 1 + \Sigma(\beta) + |\beta| *$
            $\mathtt{m} \doteq \Sigma(\gamma) + |\gamma| * \mathtt{r} \doteq |\beta| * \mathtt{stop} \doteq \mathtt{false} * \mathtt{c} \doteq -$ }
  // We can substitute for $\gamma$ and simplify (especially $\Sigma(\gamma) + |\gamma|$ to $m' + 1 + \Sigma(\gamma') + |\gamma'|$)
  { $\exists \beta, m', \gamma'.\ x \mapsto m * \mathsf{bufList}(x+1, \beta) * \mathsf{buf}(\mathtt{x}, m') * \mathsf{bufList}(\mathtt{x} + m' + 1, \gamma') *$
            $\alpha \doteq \beta \cdot (m' : \gamma') * m \doteq \Sigma(\alpha) + |\alpha| * \mathtt{x} \doteq x + 1 + \Sigma(\beta) + |\beta| *$
            $\mathtt{m} \doteq m' + 1 + \Sigma(\gamma') + |\gamma'| * \mathtt{r} \doteq |\beta| * \mathtt{stop} \doteq \mathtt{false} * \mathtt{c} \doteq -$ }
  $\mathtt{c} := [\mathtt{x}]$;
  // By proof derivation from part (2b) using the shaded predicate
  { $\exists \beta, m', \gamma'.\ x \mapsto m * \mathsf{bufList}(x+1, \beta) * \mathsf{buf}(\mathtt{x}, m') * \mathsf{bufList}(\mathtt{x} + m' + 1, \gamma') *$
            $\alpha \doteq \beta \cdot (m' : \gamma') * m \doteq \Sigma(\alpha) + |\alpha| * \mathtt{x} \doteq x + 1 + \Sigma(\beta) + |\beta| *$
            $\mathtt{m} \doteq m' + 1 + \Sigma(\gamma') + |\gamma'| * \mathtt{r} \doteq |\beta| * \mathtt{stop} \doteq \mathtt{false} * \mathtt{c} \doteq m'$ }
  if ($\mathtt{m} < \mathtt{c} + 1$) {
    { $\exists \beta, m', \gamma'.\ x \mapsto m * \mathsf{bufList}(x+1, \beta) * \mathsf{buf}(\mathtt{x}, m') * \mathsf{bufList}(\mathtt{x} + m' + 1, \gamma') *$
              $\alpha \doteq \beta \cdot (m' : \gamma') * m \doteq \Sigma(\alpha) + |\alpha| * \mathtt{x} \doteq x + 1 + \Sigma(\beta) + |\beta| *$
              $\mathtt{m} \doteq m' + 1 + \Sigma(\gamma') + |\gamma'| * \mathtt{r} \doteq |\beta| * \mathtt{stop} \doteq \mathtt{false} * \mathtt{c} \doteq m' * \mathtt{m} \dot{<} m' + 1$ }
    // Contradiction: $\mathtt{m} = m' + 1 + \Sigma(\gamma') + |\gamma'|$ and $\mathtt{m} < m' + 1$
    { False }
    $\mathtt{stop} := \mathtt{true}$
    { False }
    // From False we can get anything, so we choose to get the invariant back
    { $P_{inv}$ }
  } else {
    { $\exists \beta, m', \gamma'.\ x \mapsto m * \mathsf{bufList}(x+1, \beta) * \mathsf{buf}(\mathtt{x}, m') * \mathsf{bufList}(\mathtt{x} + m' + 1, \gamma') *$
              $\alpha \doteq \beta \cdot (m' : \gamma') * m \doteq \Sigma(\alpha) + |\alpha| * \mathtt{x} \doteq x + 1 + \Sigma(\beta) + |\beta| *$
              $\mathtt{m} \doteq m' + 1 + \Sigma(\gamma') + |\gamma'| * \mathtt{r} \doteq |\beta| * \mathtt{stop} \doteq \mathtt{false} * \mathtt{c} \doteq m' * \mathtt{m} \dot{\geq} m' + 1$ }
    $\mathtt{r} := \mathtt{r} + 1$;
    { $\exists \beta, m', \gamma'.\ x \mapsto m * \mathsf{bufList}(x+1, \beta) * \mathsf{buf}(\mathtt{x}, m') * \mathsf{bufList}(\mathtt{x} + m' + 1, \gamma') *$
              $\alpha \doteq \beta \cdot (m' : \gamma') * m \doteq \Sigma(\alpha) + |\alpha| * \mathtt{x} \doteq x + 1 + \Sigma(\beta) + |\beta| *$
              $\mathtt{m} \doteq m' + 1 + \Sigma(\gamma') + |\gamma'| * \mathtt{r} \doteq |\beta| + 1 * \mathtt{stop} \doteq \mathtt{false} * \mathtt{c} \doteq m' * \mathtt{m}' \dot{\geq} m + 1$ }
    $\mathtt{x} := \mathtt{x} + \mathtt{c} + 1$;
    // Record old value of x as $x'$
    { $\exists \beta, m', x', \gamma'.\ x \mapsto m * \mathsf{bufList}(x+1, \beta) * \mathsf{buf}(x', m') * x' \doteq x + 1 + \Sigma(\beta) + |\beta| * \mathsf{bufList}(\mathtt{x}, \gamma') *$
                $\alpha \doteq \beta \cdot (m' : \gamma') * m \doteq \Sigma(\alpha) + |\alpha| * \mathtt{x} \doteq x + 1 + \Sigma(\beta) + |\beta| + m' + 1 *$
                $\mathtt{m} \doteq m' + 1 + \Sigma(\gamma') + |\gamma'| * \mathtt{r} \doteq |\beta| + 1 * \mathtt{stop} \doteq \mathtt{false} * \mathtt{c} \doteq m' * \mathtt{m} \dot{\geq} m' + 1$ }
    $\mathtt{m} := \mathtt{m} - \mathtt{c} - 1$
    { $\exists \beta, m', x', \gamma'.\ x \mapsto m * \mathsf{bufList}(x+1, \beta) * \mathsf{buf}(x', m') * x' \doteq x + 1 + \Sigma(\beta) + |\beta| * \mathsf{bufList}(\mathtt{x}, \gamma') *$
                $\alpha \doteq \beta \cdot (m' : \gamma') * m \doteq \Sigma(\alpha) + |\alpha| * \mathtt{x} \doteq x + 1 + \Sigma(\beta) + |\beta| + m' + 1 *$
                $\mathtt{m} \doteq \Sigma(\gamma') + |\gamma'| * \mathtt{r} \doteq |\beta| + 1 * \mathtt{stop} \doteq \mathtt{false} * \mathtt{c} \doteq m'$ }
    // Entailment: appending buffer to end of buffer list
    { $\exists \beta, m', \gamma'.\ x \mapsto m * \mathsf{bufList}(x+1, \beta \cdot [m']) * \mathsf{bufList}(\mathtt{x}, \gamma') *$
              $\alpha \doteq \beta \cdot (m' : \gamma') * m \doteq \Sigma(\alpha) + |\alpha| * \mathtt{x} \doteq x + 1 + \Sigma(\beta) + |\beta| + m' + 1 *$
              $\mathtt{m} \doteq \Sigma(\gamma') + |\gamma'| * \mathtt{r} \doteq |\beta| + 1 * \mathtt{stop} \doteq \mathtt{false} * \mathtt{c} \doteq m'$ }
    // Prepare to re-establish invariant: $\beta \cdot [m']$ is the new $\beta$, $\gamma'$ is the new $\gamma$; forget actual value of c
```

```
      $\left\{\begin{array}{l} \exists \beta, m', \gamma'.\ x \mapsto m * \mathsf{bufList}(x+1, \beta \cdot [m']) * \mathsf{bufList}(\mathtt{x}, \gamma') \ * \\ \quad \alpha \doteq (\beta \cdot [m']) \cdot \gamma' * m \doteq \Sigma(\alpha) + |\alpha| * \mathtt{x} \doteq x + 1 + \Sigma(\beta \cdot [m']) + |\beta \cdot [m']| \ * \\ \quad \mathtt{m} \doteq \Sigma(\gamma') + |\gamma'| * \mathtt{r} \doteq |\beta \cdot [m']| * \mathtt{stop} \doteq \mathtt{false} * \mathtt{c} \doteq - \end{array}\right\}$
      $\{\ P_{inv}\ \}$
    }
    $\{\ P_{inv}\ \}$
  };
  $\{\ P_{inv} \land \neg(\mathtt{m} > 0 \land \mathtt{stop} = \mathtt{false})\ \}$
  // Given the invariant has $\mathtt{stop} = \mathtt{false}$, it must be that $\mathtt{m} = 0$ (cannot be $\mathtt{m} < 0$ due to $\mathtt{m} \in \mathbb{N}$)
  $\left\{\begin{array}{l} \exists \beta, \gamma.\ x \mapsto m * \mathsf{bufList}(x+1, \beta) * \mathsf{bufList}(\mathtt{x}, \gamma) * \alpha \doteq \beta \cdot \gamma \ * \\ \quad m \doteq \Sigma(\alpha) + |\alpha| * \mathtt{m} \doteq \Sigma(\gamma) + |\gamma| * \mathtt{r} \doteq |\beta| * \mathtt{stop} \doteq \mathtt{false} * \mathtt{c} \doteq - * \mathtt{m} \doteq 0 \end{array}\right\}$
  // $\mathtt{m} = 0$ means $\gamma = \epsilon$ and $\beta = \alpha$; $\mathsf{bufList}(\mathtt{x}, \epsilon)$ unfolds to $\mathtt{x} \mathrel{\dot\in} \mathbb{N}$ and can be forgotten, as can stop, c, and m
  $\{\ x \mapsto m * \mathsf{bufList}(x+1, \alpha) * m \doteq \Sigma(\alpha) + |\alpha| * \mathtt{r} \doteq |\alpha|\ \}$
  // Entailment (E3) on the shaded bit
  $\{\ x \mapsto m * \circledast_{0 \leq i < \Sigma(\alpha)+|\alpha|}(x+1+i \mapsto -) * m \doteq \Sigma(\alpha) + |\alpha| * \mathtt{r} \doteq |\alpha|\ \}$
  // Shift indexes of the iterated star and fold buf
  $\{\ \mathsf{buf}(x, \Sigma(\alpha) + |\alpha|) * \mathtt{r} \doteq |\alpha|\ \}$
  return r
}
$\{\ \mathsf{buf}(x, \Sigma(\alpha) + |\alpha|) * \mathtt{ret} \doteq |\alpha|\ \}$
```

Marks breakdown: 1 mark for entering and exiting the function correctly, 1 mark for establishing invariant, 3 marks for the non-visited branch of the if, 2 marks for recognising the shaded entailment on appending a buffer to a buffer list, 1 mark for re-establishing invariant, 1 mark for using entailment (E3), 3 marks for the rest.

Marks: **12**

The three parts carry, respectively, 30%, 10%, and 60% of the marks.