COMP 2401 B

Test #2 (version 1)

1. [2 marks] b
2. [2 marks] c
3. [2 marks] d
4. [2 marks] a
5. [2 marks] a
6. [2 marks] c

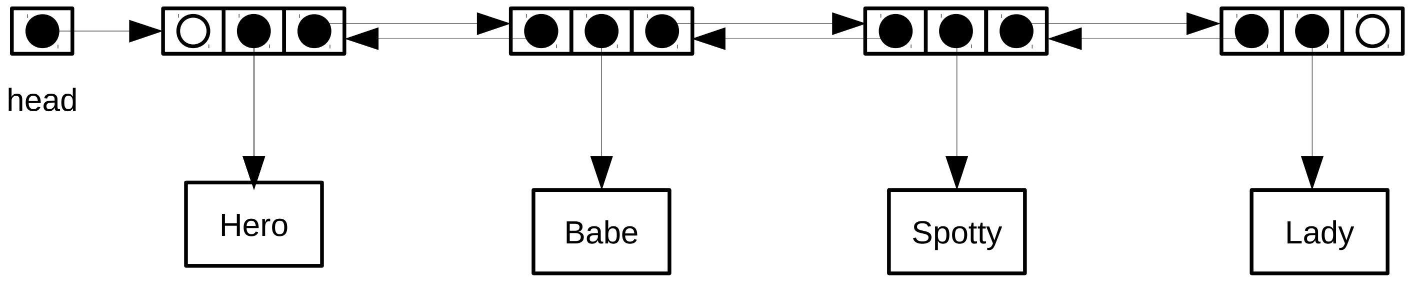
7. [10 marks]

```
void initChicken(char *n, int r, ChickenType **chick) {
  *chick = malloc(sizeof(ChickenType));
  strcpy((*chick)->name, n);
  (*chick)->rank = r;
}
int main()
{
  ChickenType *newChick;
  initChicken("Gertrude", 3, & newChick);
  printf("Name is %s, rank is %d\n", newChick->name, newChick->rank);
  free(newChick);
}
```

Marking:
-- 2 marks for making parameter a double pointer in `initChicken()`
-- 2 marks for allocating `ChickenType` in `initChicken()`
-- 2 marks for dereferencing `chick` in `initChicken()` (1 mark each)
-- 2 marks for passing address of `newChick` to `initChicken()`
-- 2 marks for freeing `newChick`

8. [28 marks]

a. [6 marks]



Marking:
-- 1 mark for correct pointer to head node
-- 1 mark for first node's prev set to null
-- 1 mark for last node's next set to null
-- 1 mark for 3 next pointers
-- 1 mark for 3 prev pointers
-- 1 mark for correct pointers to data structures, in correct order

b. [10 marks]

```
    NodeType *newNode;

  // 4 marks for allocating and initializing node
  // -- 2 marks for malloc (zero if freed)
  // -- 2 marks for initializing node data and prev
    newNode = (NodeType *) malloc(sizeof(NodeType));
    newNode->data = newAnimal;
    newNode->prev = NULL;

  // 2 marks for setting new node's next to head
    newNode->next = list->head;

  // 2 marks for checking that old head is not null
  // and setting old head's prev to new node
    if (list->head != NULL)
      list->head->prev = newNode;

  // 2 marks for setting new head
    list->head = newNode;
```

c. [12 marks]

```
    NodeType   *currNode;
    NodeType   *lastNode;
    AnimalType *goner;

  // 2 marks for dealing with empty list case
    if (list->head == NULL)
      return 0;

  // 2 marks for correctly looping through list
  // 2 marks for saving last node
    currNode = list->head;
    lastNode = NULL;
    while (currNode != NULL) {
      lastNode = currNode;
      currNode = currNode->next;
    }

  // 1 mark for checking that last node has a prev; if so:
  // 1 mark for setting last node's prev node's next to NULL
    if (lastNode->prev != NULL)
      lastNode->prev->next = NULL;

  // 1 mark for saving last node's data
    goner = lastNode->data;

  // 2 marks for freeing last node
    free(lastNode);

  // 1 mark for returning last node's data
    return goner;
```