1. We want to add an inventory system to a text adventure game so that the player can collect items. First, we’ll implement a bag data structure that holds items in a linked list. Each item_t has an associated weight, and each bag_t has a max_weight that determines its holding capacity (see the definitions below). In the left text area for item_node_t, define the necessary data type to serve as the nodes in a linked list of items, and in the right text area, add any necessary fields to the bag_t definition.

```c
typedef struct item {
    int weight;
    // other fields not shown
} item_t;

typedef struct item_node {
    // (a) FILL IN HERE
} item_node_t;

typedef struct bag {
    int max_weight;
    int current_weight;
    // add other fields necessary
    // (b) FILL IN HERE
} bag_t;
```

c) Complete the add_item() function, which should add item into bag only if adding the item would not cause the weight of the bag contents to exceed the bag’s max_weight. The function should return 0 if the item could not be added, or 1 if it succeeded. Be sure to update the bag’s current_weight. You do not need to check if malloc() returns NULL. Insert the new item into the list wherever you wish.

```c
int add_item(item_t *item, bag_t *bag) {
    if (_________________________________________) {
        return 0;
    }
    item_node_t *new_node = ___________________________________________
    // Add more code below...
    return 1;
}
```

(d) Finally, we want an empty_bag() function that frees the bag’s linked list but NOT the memory of the items themselves and NOT the bag itself. The bag should then be “reset”, ready for add_item. Assume that the operating system immediately fills any freed memory with garbage. Fill in the functions below.
void empty_bag(bag_t *bag) {
    free_contents(__________);
    // FILL IN HERE
}

void free_contents(__________)
{ // FILL IN HERE
}

(e) Now suppose that we care about the order of items in our bag. However, because we’re clumsy, the only possible way for us to rearrange items is to reverse their order in the list.

void reverse_list(bag_t *bag) {

}

2. Questions (a) and (b) refer to the C code to the right.

a) In which memory sections (code, static, heap, stack) do the following reside?

arg ___________ arr ___________ *str ___________
*val ___________ (bonus) *arr ___________

b) Name a C operation that would treat arr and ptr differently:
________________________________________________________________________

c) Which of the best-, first-, next-fit schemes would succeed for all 5 of the following sequence of malloc and free requests on a malloc-able region of memory only 8 bytes long? Circle those that would and show the resulting contents of memory for each one. E.g., After the “a=malloc(4)” call, all schemes should have the leftmost 4 boxes labeled “a”.

```
a = malloc(4); b = malloc(1); free(a); c = malloc(3); d = malloc(4);
```

<table>
<thead>
<tr>
<th>best-fit</th>
<th>first-fit</th>
<th>next-fit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>