1 Advanced C

Suppose we’ve defined a linked list struct as follows. Assume *lst points to the first element of the list, or is NULL if the list is empty.

```c
struct ll_node {
    int first;
    struct ll_node* rest;
}
```

1.1 Implement prepend, which adds one new value to the front of the linked list. Hint: why use ll_node ** lst instead of ll_node * lst?

```c
void prepend(struct ll_node** lst, int value)
```

1.2 Implement free_ll, which frees all the memory consumed by the linked list.

```c
void free_ll(struct ll_node** lst)
```

2 Memory Management

2.1 For each part, choose one or more of the following memory segments where the data could be located: code, static, heap, stack.

(a) Static variables

(b) Local variables

(c) Global variables
(d) Constants
(e) Machine Instructions
(f) Result of malloc
(g) String Literals

2.2 Write the code necessary to allocate memory on the heap in the following scenarios

(a) An array arr of k integers
(b) A string str containing p characters
(c) An n × m matrix mat of integers initialized to zero.

2.3 What is wrong with the C code below?

```c
1 int* pi = malloc(314 * sizeof(int));
2 if (!raspberry) {
3    pi = malloc(1 * sizeof(int));
4 }
5 return pi;
```