Bonus Part for Machine Problem 1

I will explain some more details and also how free_data_pointer works for the bonus part.

The 4 Pointers (free_data_pointer is only for the bonus part)

* head_pointer: always point to the allocated address from malloc, you should free it in the Destroy function if you allocate the initial memory to it.

* front_pointer: always point to the first node in the linked list; if the first node is deleted, it will point to the second node if there is a second one. it is set to null when the linked list is or becomes empty;

* free_pointer: always point to the last initialized node in the linked list; it is set to null when the linked list is empty or becomes empty.

*free_data_pointer: always point to the address the next insertion will happen. The introduction of this pointer actually saves the memory from the deleted node, you will see examples.

free_data_pointer actually controls the “free list” in which we will insert new node. It effectively uses the space from delated node.

The 4 Functions

- I wrote some pseudocode, they reflect the basic idea, but they are not guaranteed to compile and run.
- Init(int M, int b){
  - Allocate memory by using malloc();
  - Construct nodes and connect them by using the “connector” *next:
    - you should start from the given memory address given by malloc, remember to do a typecasting if you use a node* type iterator, eg. node* node_iterator = (node*)malloc(M);
    - Since you know how many nodes you will create, so you can use a while loop or for loop until you have initialized M/b nodes;
    - Last node should point to a NULL
    - Here is an simple example, assuming we haven’t reached the last node.

```
while we haven’t finished initialization for all nodes{
    if you are working with the last node, then its next will be NULL and break the   while loop here.
    node_iterator ->key = dummy_value;
    node_iterator->value_len = dummy_value;
    node* next_node = (node*)((char*)node_iterator + b);
    node_iterator->next = next_node;
    node_iterator = node_iterator->next;
}
```

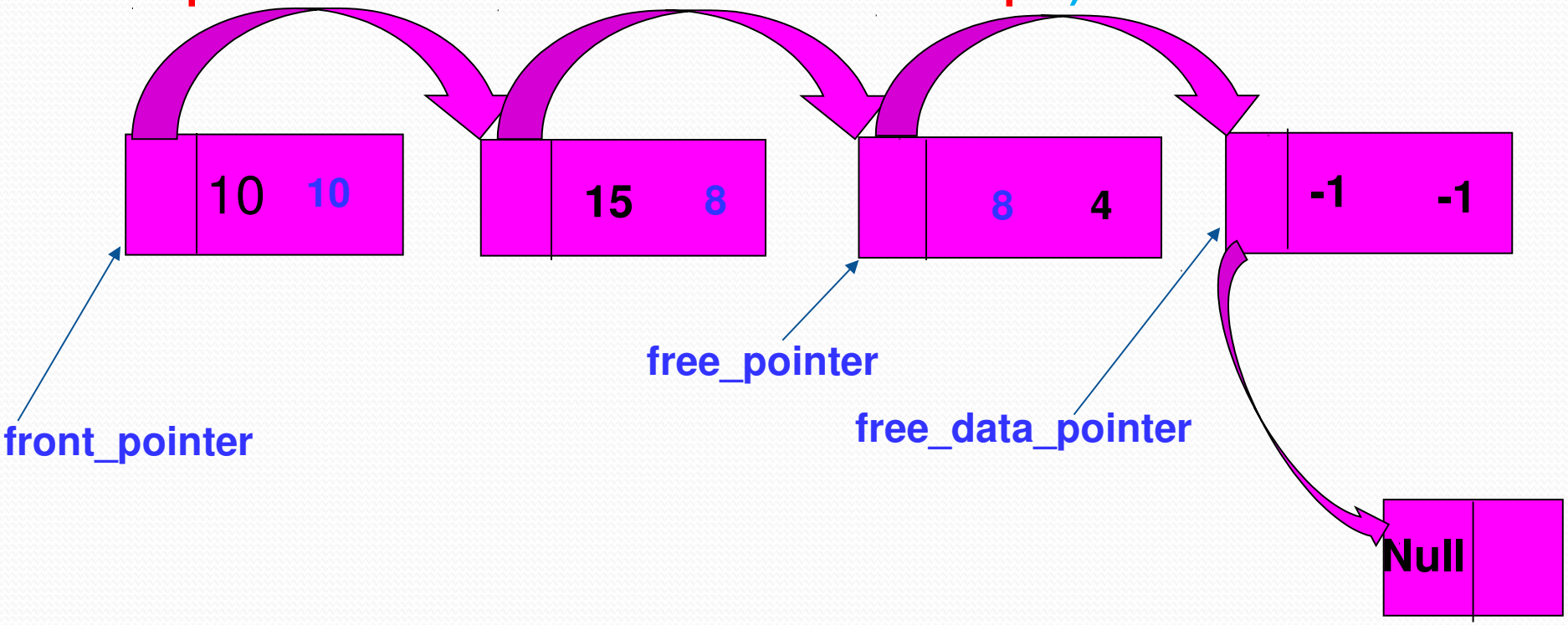
    - Remember to set initial values for those linked list class members

Insert(key, data, val_len) for Bonus Part

- Constraints for insertion:
  - Is the linked list already full?
  - Is the val_len is bigger than the maximum_data_size (block_size - sizeof(node));
  - Is the key already used for some node?
- Since free_data_pointer points to the node we are going to insert next, thus when a (key, data, val_len) comes, you can set key and val_len for the node the free_data_pointer points to. Use memcpy(*des, *src, amount) to make a copy of the src to des, des is the starting address of payload part. Note that if you do not use free_data_pointer, it will be free_pointer+b+sizeof(node); if you use free_data_pointer, it should be free_data_pointer+ sizeof(node);
- When a new node is inserted, the free_pointer points to the new node. And free_data_pointer points to the next location the future node will be inserted at.

Following Slides give you a simple example how the free_data_pointer works here.

We have a linked list like the following, each node has two values, key and value_len. The linked list has 4 blocks, but we only insert 3 pairs in this example. For now, you can see where the pointers are. (Note, in the project, the memory we allocated is contiguous, here, I separated them to make these slides simple)



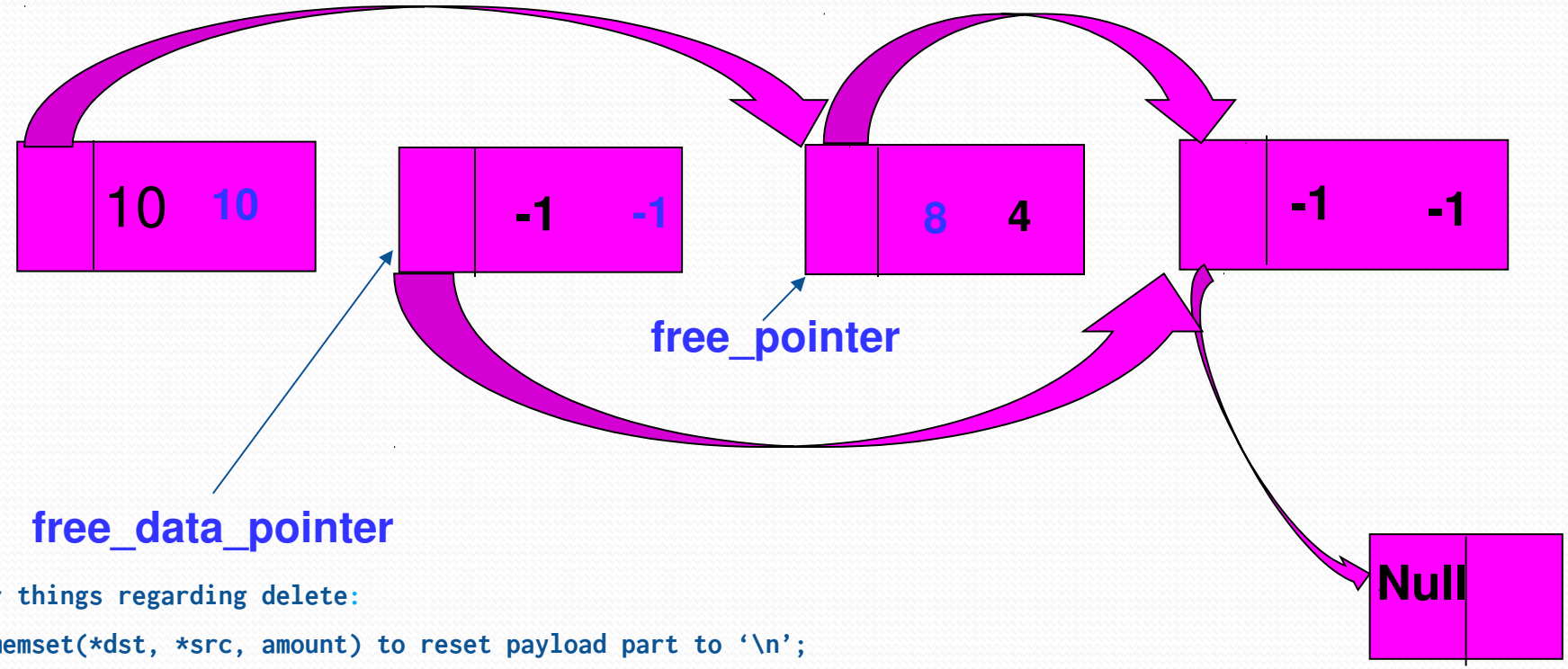
Now, lets use Delete function like this: linked_list.Delete(15);

After successfully deletion, the linked list should be like the following

For coding this behavior of free_data_pointer, you just need to do (assuming you already found the node you want to delete): delete_node,

delete_node->next = free_data_pointer;

Delete_node = free_data_pointer; //**make sure you break the connection between delete_node and its previous one and its next one. Also connect its previous one and its next one.**



Other things regarding delete:

Use memset(*dst, *src, amount) to reset payload part to '\n';

Also reset the key and val_len of the deleted node to the initial value set in Init

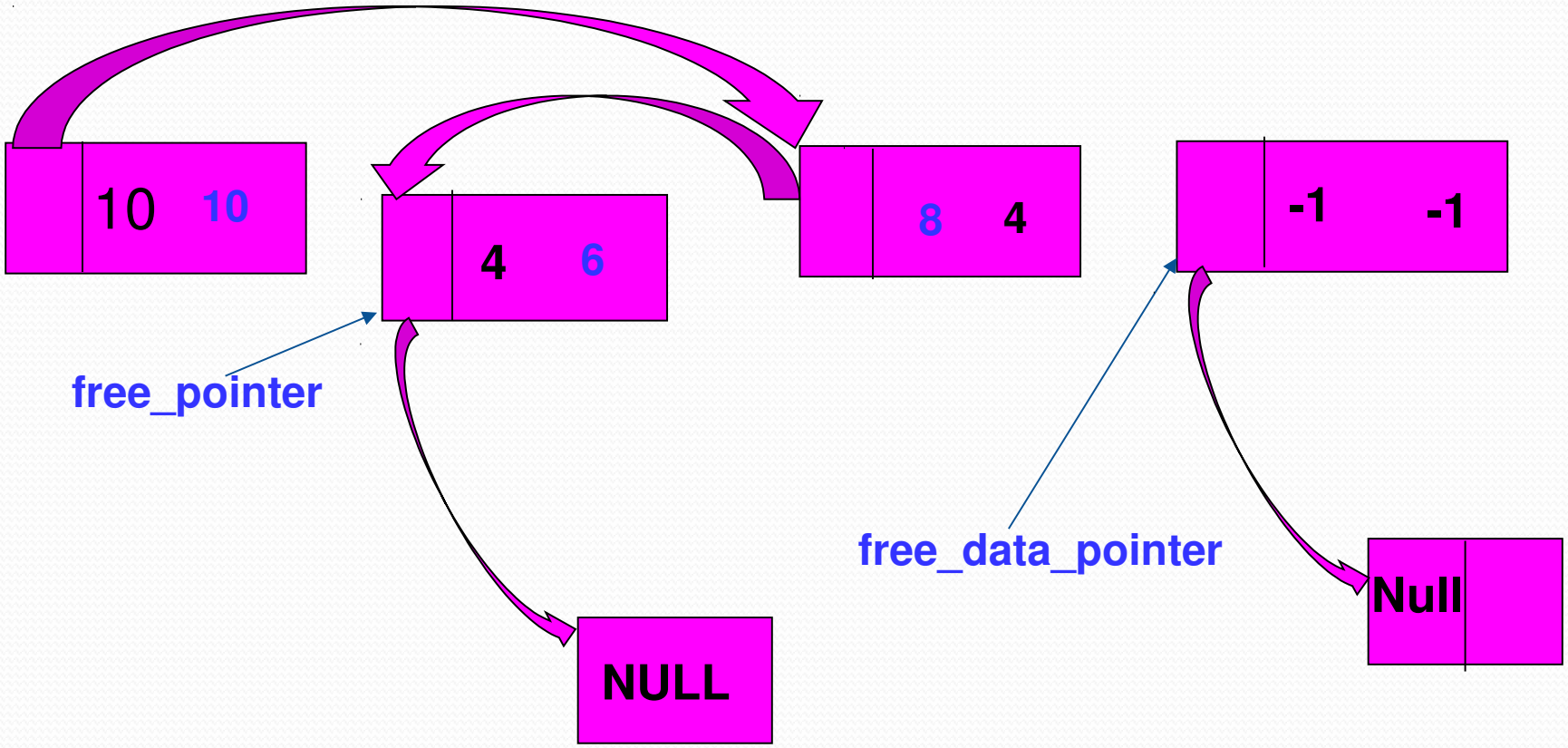
Now, lets use another Insert function like this: linked_list.Insert(4, “haha”, 6);

After successfully insertion, the linked list should be like this:

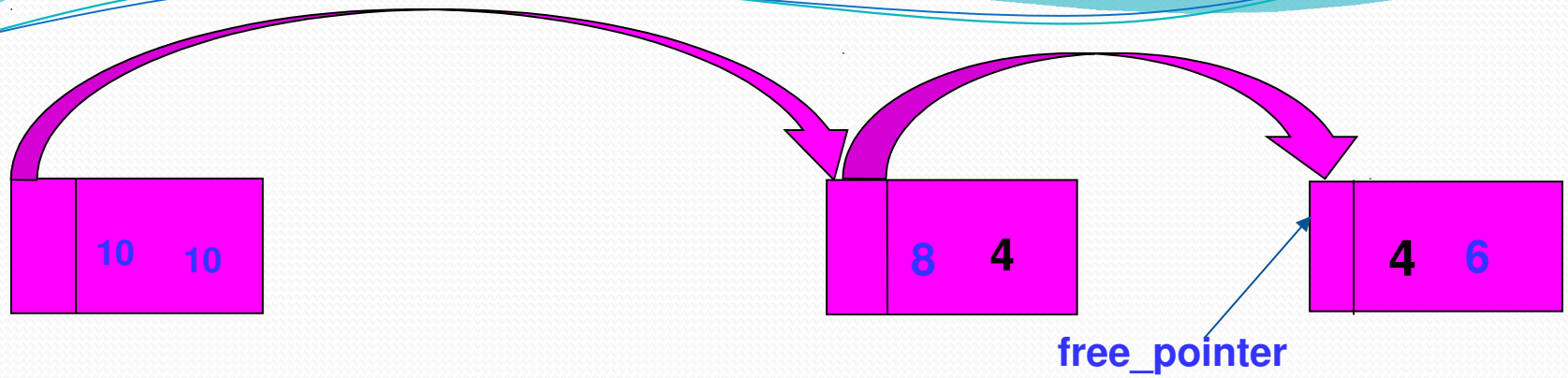
For coding this behavior of free_data_pointer, you just need to do (assuming you already found the node you want to delete): delete_node,

delete_node->next = free_data_pointer;

Delete_node = free_data_pointer; //**make sure you break the connection between delete_node and its previous one and its next one. Also connect its previous one and its next one.**



Without free_data_pointer VS With free_data_pointer:



The one with free_data_pointer effectively use the space of the deleted node.