Append one list to the end of another

```
void append_second_list(struct node** a, struct node** b);
```

Note that both lists are referenced by a pointer to a pointer. The heads of both lists might be updated (
1. List “a”’s head might change if it was initially a 0 element list
2. List “b”’s head will be assigned to null.

Example:

If “a” and “b” look like this 1->5->7->NULL and 4->6->8->NULL the list “a” should point to 1->5->7>4->6->8->NULL

APPEND ONE LIST TO THE END OF ANOTHER

```
void append_second_list(struct node** list_a, struct node** list_b) {
  assert(list_a != NULL && list_b != NULL);

  if (*list_a == NULL) {
    *list_a = *list_b;
    *list_b = NULL;
    return;
  }

  struct node* head = *list_a;
  while (head->next != NULL) {
    head = head->next;
  }

  head->next = *list_b;
  *list_b = NULL;
}
```

IF LIST A IS AN EMPTY LIST, SIMPLY POINT TO LIST B AND RETURN

WALK TO THE END OF LIST A, TILL HEAD POINTS TO THE LAST ELEMENT IN THE LIST

POINT THE LAST ELEMENT OF LIST A TO THE FIRST ELEMENT OF LIST B

Front back split

```
void front_back_split(struct node* source, struct node** frontRef, struct node**
                                                                    backRef);
```

Examples:

1->2->3->4->5->NULL becomes 1->2->3->NULL 4->5->NULL
1->2->3->4->NULL becomes 1->2->NULL 3->4->NULL

The extra element for odd numbered lists is appended to the first list.

Potential solutions

1. Can traverse the entire length of the list twice, once to get the length of the list and the second time to split it. This is sub-optimal, try solving it by traversing the list just once.

HINT: You can use the 2 pointer solution, a “fast” pointer which moves 2 elements in every iteration and a “slow” pointer which moves just once in an iteration

This is remarkably complicated to get right for all lengths of a list, so don’t worry if you struggle!

FRONT BACK SPLIT

```
void front_back_split(struct node* source,
                      struct node** frontRef,
                      struct node** backRef) {
  assert(frontRef != NULL &&  backRef != NULL);

  if (source == NULL) {
    *frontRef = NULL;
    *backRef = NULL;
    return;
  }

  if (source->next == NULL) {
    *frontRef = source;
    *backRef = NULL;
    return;
  }

  struct node* slow = source;
  struct node* fast = source;
  while (fast != NULL) {
    fast = fast->next;
    if (fast == NULL) {
      break;
    }
    fast = fast->next;
    if (fast != NULL) {
      slow = slow->next;
    }
  }

  *frontRef = source;
  *backRef = slow->next;
  slow->next = NULL;
}
```

ALWAYS CHECK THE EDGE CASES FOR NULLS ETC.

ADDING IN A SPECIAL CHECK FOR A SINGLE ELEMENT LIST, THE ONE ELEMENT GOES TO THE FIRST LIST

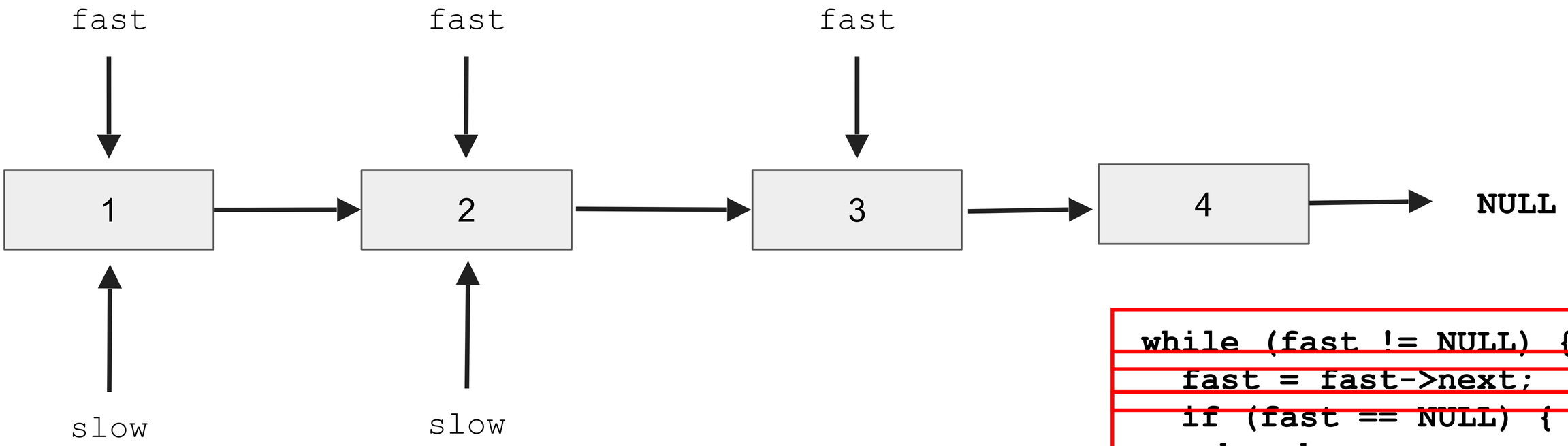
BOTH SLOW AND FAST POINTERS START AT THE HEAD

THE FAST POINTER MOVES TWICE IN AN ITERATION

THE SLOW POINTER MOVES JUST ONCE

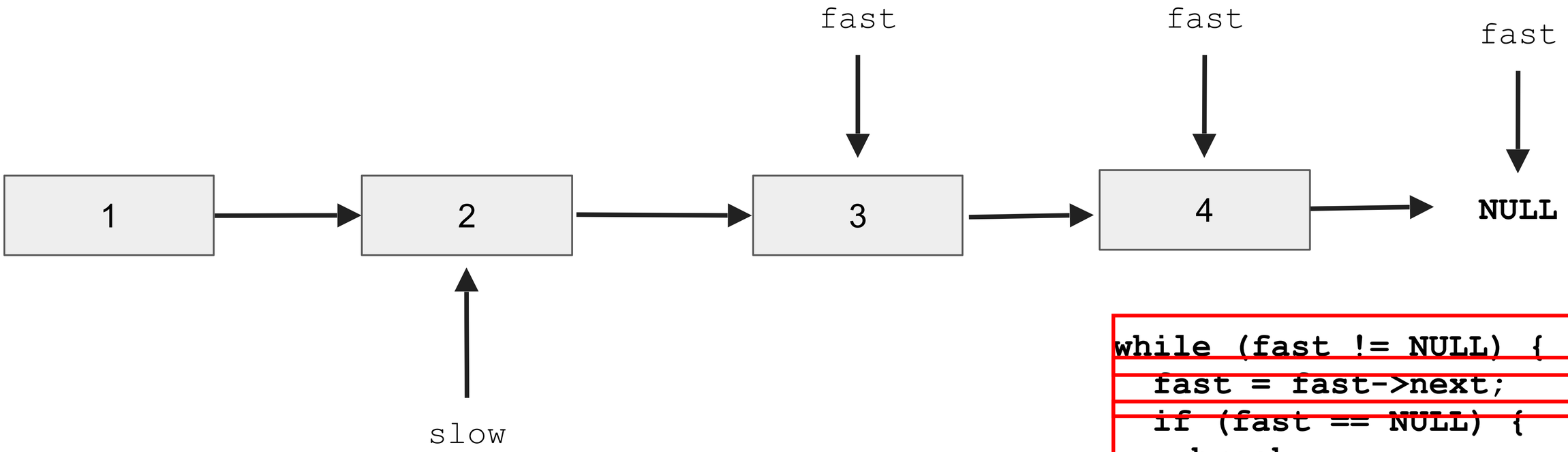
THE SLOW POINTER STOPS AT THE LAST ELEMENT OF THE FIRST LIST, SET ITS NEXT VALUE TO NULL TO TERMINATE THE FIRST LIST

FRONT BACK SPLIT



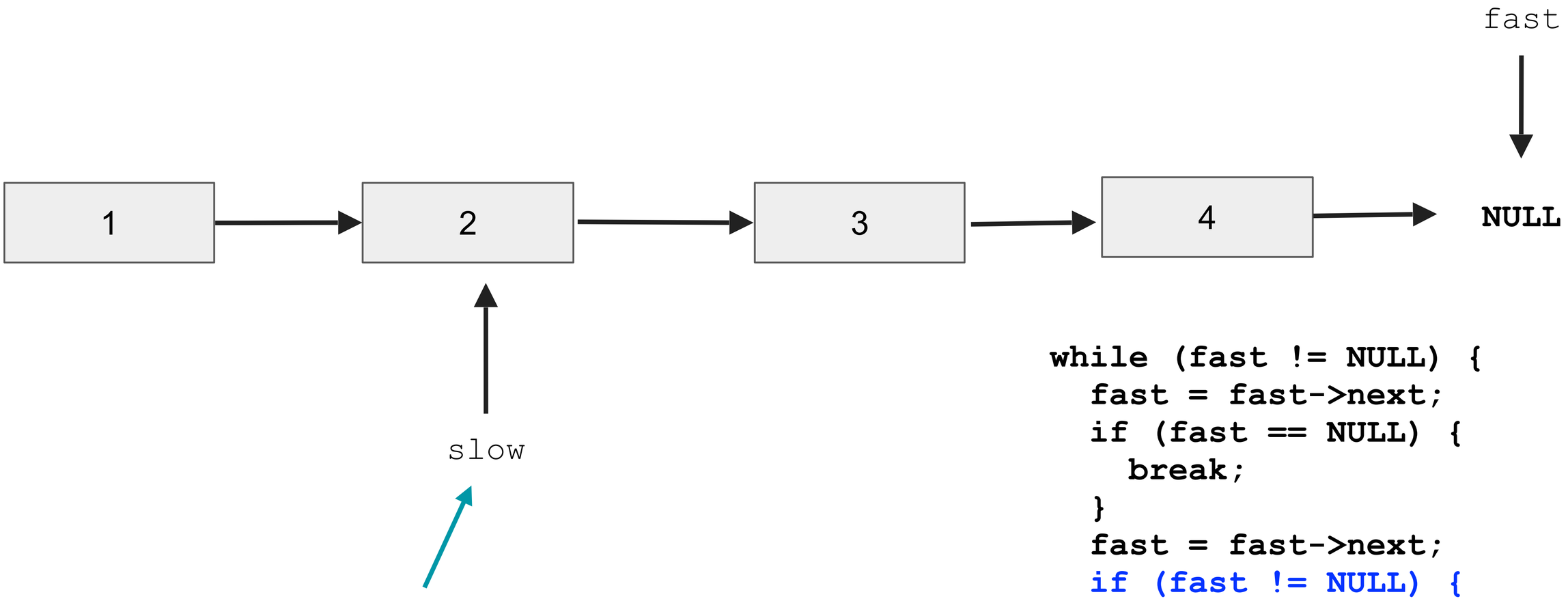

```
while (fast != NULL) {
  fast = fast->next;
  if (fast == NULL) {
    break;
  }
  fast = fast->next;
  if (fast != NULL) {
    slow = slow->next;
  }
}
```

FRONT BACK SPLIT - END OF ITERATION 1




```
while (fast != NULL) {
  fast = fast->next;
  if (fast == NULL) {
    break;
  }
  fast = fast->next;
  if (fast != NULL) {
    slow = slow->next;
  }
}
```

FRONT BACK SPLIT - END OF ITERATION 2




```
while (fast != NULL) {
  fast = fast->next;
  if (fast == NULL) {
    break;
  }
  fast = fast->next;
  if (fast != NULL) {
    slow = slow->next;
  }
}
```

THE SLOW POINTER STAYS, AS FAST HAS REACHED THE END OF THE LIST

FRONT BACK SPLIT

```
void front_back_split(struct node* source,
                      struct node** frontRef,
                      struct node** backRef) {
  assert(frontRef != NULL &&  backRef != NULL);

  if (source == NULL) {
    *frontRef = NULL;
    *backRef = NULL;
    return;
  }

  if (source->next == NULL) {
    *frontRef = source;
    *backRef = NULL;
    return;
  }

  struct node* slow = source;
  struct node* fast = source;
  while (fast != NULL) {
    fast = fast->next;
    if (fast == NULL) {
      break;
    }
    fast = fast->next;
    if (fast != NULL) {
      slow = slow->next;
    }
  }

  *frontRef = source;
  *backRef = slow->next;
  slow->next = NULL;
}
```

ALWAYS CHECK THE EDGE CASES FOR NULLS ETC.

ADDING IN A SPECIAL CHECK FOR A SINGLE ELEMENT LIST, THE ONE ELEMENT GOES TO THE FIRST LIST

BOTH SLOW AND FAST POINTERS START AT THE HEAD

THE FAST POINTER MOVES TWICE IN AN ITERATION

THE SLOW POINTER MOVES JUST ONCE

THE SLOW POINTER STOPS AT THE LAST ELEMENT OF THE FIRST LIST, SET ITS NEXT VALUE TO NULL TO TERMINATE THE FIRST LIST