Thread 1

```
int list remove front(list *lp) {
    int num;
    list element *element;
    assert(!is empty(lp));
    element = lp->first;
    num = Ip->first->item;
    if (lp->first == lp->last) {
        lp->first = lp->last = NULL;
    } else {
        lp->first = element->next;
    lp->num in list--;
    free(element);
    return num;
```

```
int list remove front(list *lp) {
    int num:
    list element *element;
    assert(!is empty(lp));
    element = lp->first;
    num = lp->first->item;
    if (lp->first == lp->last) {
        lp->first = lp->last = NULL;
    } else {
         lp->first = element->next;
    Ip->num in list--;
    free(element);
    return num;
```

```
int list remove front(list *lp) {
         int num:
         list element *element;
1
         assert(!is_empty(lp));
         element = lp->first;
         num = Ip->first->item;
         if (lp->first == lp->last) {
              lp->first = lp->last = NULL;
         } else {
              lp->first = element->next;
          lp->num in list--;
         free(element);
         return num;
```

```
int list remove front(list *lp) {
    int num:
    list element *element;
    assert(!is_empty(lp));
    element = lp->first;
    num = lp->first->item;
    if (lp->first == lp->last) {
        lp->first = lp->last = NULL;
    } else {
         lp->first = element->next;
    Ip->num in list--;
    free(element);
    return num;
```

```
int list remove front(list *lp) {
         int num:
         list element *element;
1
         assert(!is_empty(lp));
         element = lp->first;
         num = Ip->first->item;
         if (lp->first == lp->last) {
              lp->first = lp->last = NULL;
         } else {
              lp->first = element->next;
         lp->num in list--;
         free(element);
         return num;
```

```
int list remove front(list *lp) {
    int num:
    list element *element;
    assert(!is_empty(lp));
    element = lp->first;
    num = lp->first->item;
    if (lp->first == lp->last) {
        lp->first = lp->last = NULL;
    } else {
         lp->first = element->next;
    Ip->num in list--;
    free(element);
    return num;
```

```
Thread 2
```

```
int list remove front(list *lp) {
                                                  int list remove front(list *lp) {
         int num:
                                                       int num:
         list element *element;
                                                       list element *element;
1
         assert(!is_empty(lp));
                                                       assert(!is_empty(lp));
         element = lp->first;
                                                       element = lp->first;
                                                       num = Ip->first->item;
         num = lp->first->item;
         if (lp->first == lp->last) {
                                                       if (lp->first == lp->last) {
              lp->first = lp->last = NULL;
                                                           lp->first = lp->last = NULL;
                                                       } else {
         } else {
              lp->first = element->next;
                                                           lp->first = element->next;
3
                                                       Ip->num in list--;
          lp->num in list--;
         free(element);
                                                       free(element);
         return num;
                                                       return num;
```

Thread 1

```
int list remove front(list *lp) {
         int num:
         list element *element;
1
         assert(!is_empty(lp));
         element = lp->first;
         num = lp->first->item;
         if (lp->first == lp->last) {
              lp->first = lp->last = NULL;
         } else {
              lp->first = element->next;
3
         lp->num in list--;
         free(element);
         return num;
```

```
int list remove front(list *lp) {
    int num:
    list element *element;
    assert(!is_empty(lp));
    element = lp->first;
    num = lp->first->item;
    if (lp->first == lp->last) {
        lp->first = lp->last = NULL;
    } else {
         lp->first = element->next;
    Ip->num in list--;
    free(element);
    return num;
```

- 1. Both threads return the same value
- 2. Decrements num_in_list twice even though only one item has been removed
- 3. Dereferences an element that has been freed
- 4. Calls free() twice on the same element

Assume there is one item in the list.

Thread 1

```
int list remove front(list *lp) {
    int num:
    list element *element;
    assert(!is empty(lp));
    element = lp->first;
    num = lp->first->item;
    if (lp->first == lp->last) {
        lp->first = lp->last = NULL;
    } else {
        lp->first = element->next;
    lp->num in list--;
    free(element);
    return num;
```

```
void list append(list *lp, int new item) {
    list element *element =
        malloc(sizeof(list element));
    element->item = new item;
    assert(!is in list(lp, new item));
    if (is empty(lp)) {
        lp->first = element;
        lp->last = element;
    } else {
        lp->last->next = element;
        lp->last = element;
    lp->num in list++;
```

```
int list remove front(list *lp) {
    int num:
    list element *element;
    assert(!is_empty(lp));
    element = lp->first;
    num = lp->first->item;
    if (lp->first == lp->last) {
        lp->first = lp->last = NULL;
    } else {
        lp->first = element->next;
    lp->num in list--;
    free(element);
    return num;
```

```
void list append(list *lp, int new item) {
    list element *element =
        malloc(sizeof(list element));
    element->item = new item;
    assert(!is in _list(lp, new item));
    if (is empty(lp)) {
        lp->first = element;
        lp->last = element;
    } else {
        lp->last->next = element;
        lp->last = element;
    lp->num in list++;
```

```
int list remove front(list *lp) {
                                            void list append(list *lp, int new item) {
                                                list element *element =
    int num:
    list element *element;
                                                     malloc(sizeof(list element));
    assert(!is_empty(lp));
                                                element->item = new item;
    element = lp->first;
                                                assert(!is in list(lp, new item));
    num = lp->first->item;
                                                if (is empty(lp)) {
    if (lp->first == lp->last) {
                                                     lp->first = element;
        lp->first = lp->last = NULL;
                                                     lp->last = element;
    } else {
                                                } else {
        lp->first = element->next;
                                                     lp->last->next = element;
                                                    lp->last = element;
    lp->num in list--;
    free(element);
                                                lp->num in list++;
    return num;
```

```
int list remove front(list *lp) {
                                            void list append(list *lp, int new item) {
                                                list element *element =
    int num:
    list element *element;
                                                    malloc(sizeof(list element));
    assert(!is empty(lp));
                                                element->item = new item;
    element = lp->first;
                                                assert(!is in list(lp, new item));
    num = lp->first->item;
                                                if (is empty(lp)) {
    if (lp->first == lp->last) {
                                                    lp->first = element;
                                                    lp->last = element;
        lp->first = lp->last = NULL;
                                                } else {
    } else {
        lp->first = element->next;
                                                    lp->last->next = element;
                                                    lp->last = element;
    lp->num in list--;
                                                lp->num in list++;
    free(element);
    return num;
```

Thread 1 removes the only item in the list after thread 2 checks if the list is empty.

As a result, thread 2 will try to dereference a NULL pointer (lp->last) when it resumes.