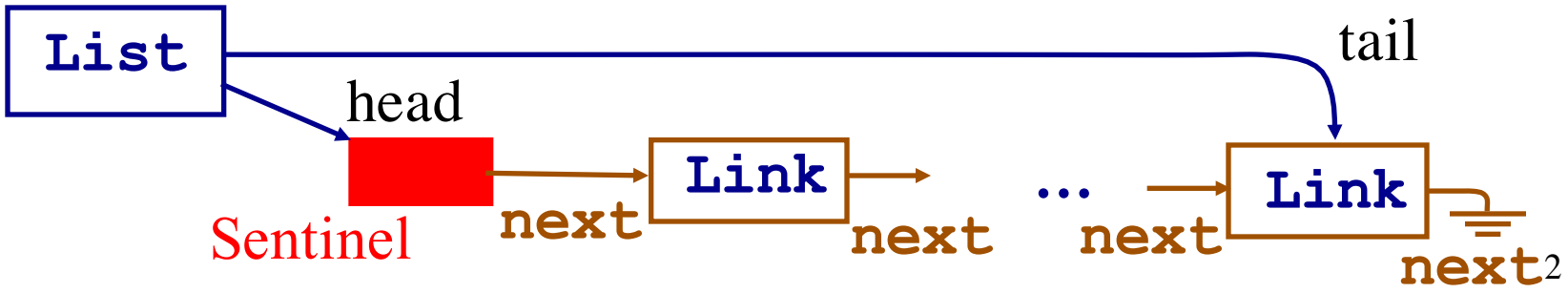
CS 261: Data Structures

List Queue

List Deque

List Queue

- Add elements to the tail of the list
- Remove elements from the head of the list
- Why not the other way around?



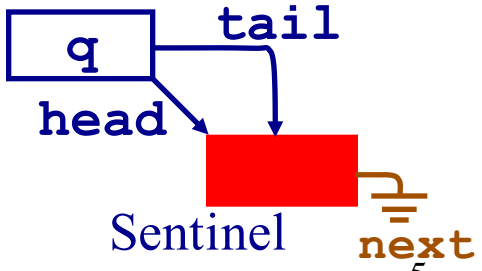
Implementation of ListQueue

Structure for List Queue

```
struct Link {
    TYPE  value;
    struct Link * next;
};

struct ListQueue {
    struct Link * head;
    struct Link * tail;
};
```

Init Makes the Sentinel

```
void initListQueue (struct ListQueue *q){
    struct Link *sentinel =
          (struct Link *)malloc(sizeof(struct Link));
    assert(sentinel != 0);
    sentinel->next = 0;
    q->head = q->tail = sentinel;
}
```

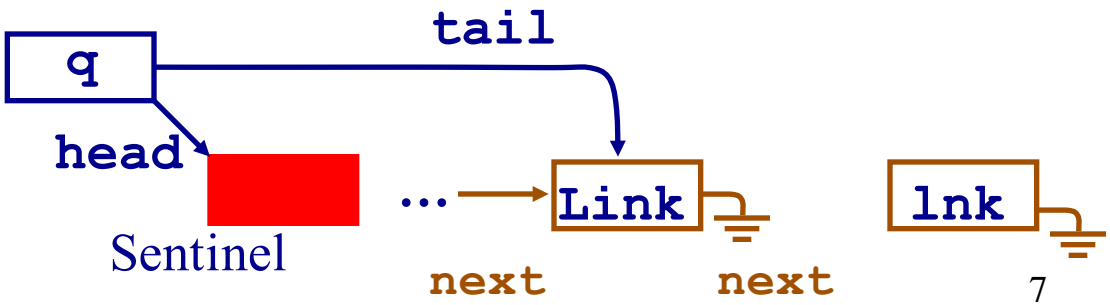


isEmpty

```
int isEmptyListQueue (struct ListQueue *q) {

    return q->head == q->tail;

}
```

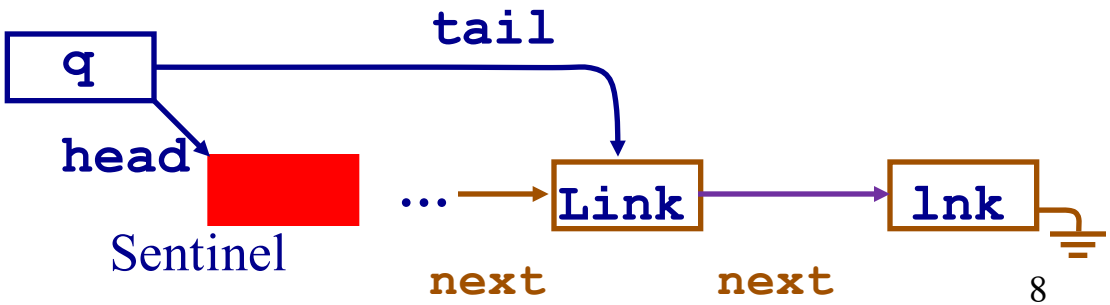
Elements are Added to Tail

```
void addBackListQueue (struct ListQueue *q, TYPE val)
{
   struct Link * lnk= (struct Link *)
                          malloc(sizeof(struct Link));
   assert(lnk != 0);
   lnk->next = 0;
   lnk->value = val;


}
```



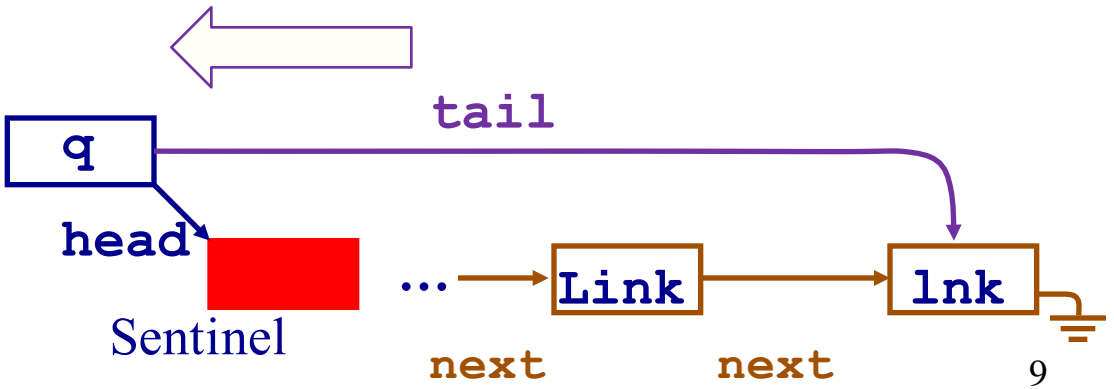
Add an Element (to Tail)

```
void addBackListQueue (struct ListQueue *q, TYPE val)
{
   struct Link * lnk= (struct Link *)
                          malloc(sizeof(struct Link));
   assert(lnk != 0);
   lnk->next = 0;
   lnk->value = val;
   q->tail->next = lnk;

}
```



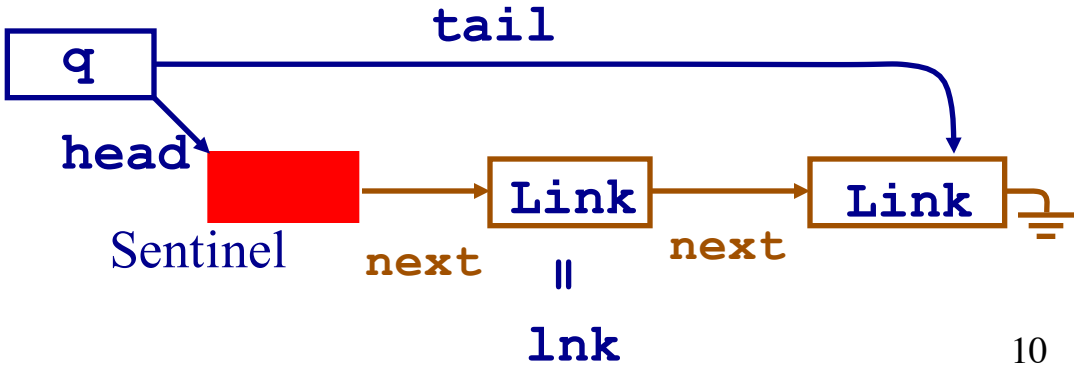
Add

```
void addBackListQueue (struct ListQueue *q, TYPE val)
{
   struct Link * lnk= (struct Link *)
                          malloc(sizeof(struct Link));
   assert(lnk != 0);
   lnk->next = 0;
   lnk->value = val;
   q->tail->next = lnk;
   q->tail = lnk;
}
```



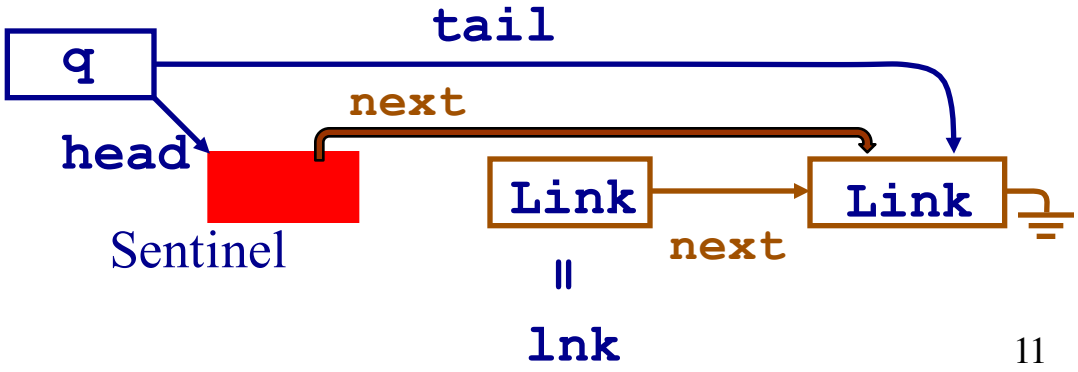
Remove (from Front)

```
void removeListQueue (struct listQueue *q) {

    struct link * lnk = q->head->next;




}
```



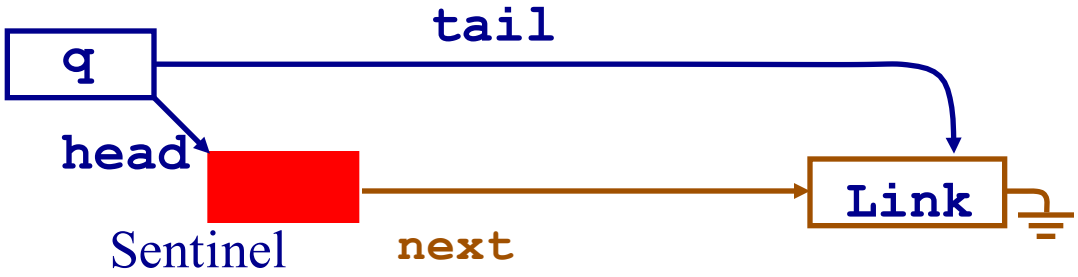
Remove

```
void removeListQueue (struct listQueue *q) {
    struct link * lnk = q->head->next;
    assert ( ! isEmptyListQueue(q));
    q->head->next = lnk->next;


}
```



Remove

```
void removeListQueue (struct listQueue *q) {
    struct link * lnk = q->head->next;
    assert ( ! isEmptyListQueue(q));
    q->head->next = lnk->next;
    if(q->head->next == 0) q->tail = q->head;
    free (lnk);
}
```

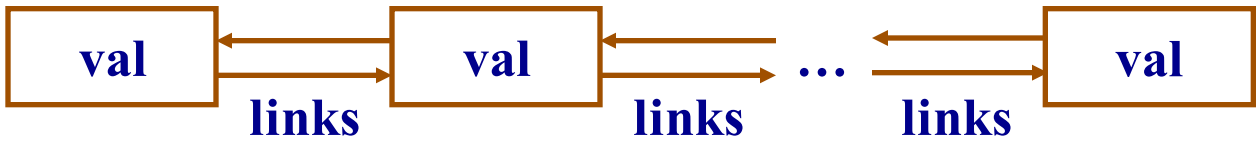


Doubly Linked Lists

Double Links

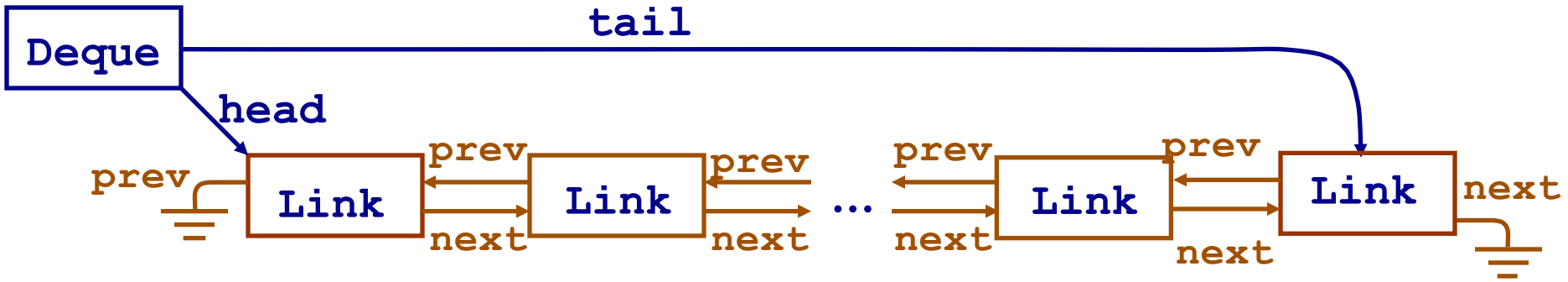
- Allow access to both next and previous link

```
struct dlink {
    TYPE value;
    struct dlink * next;
    struct dlink * previous;
};
```



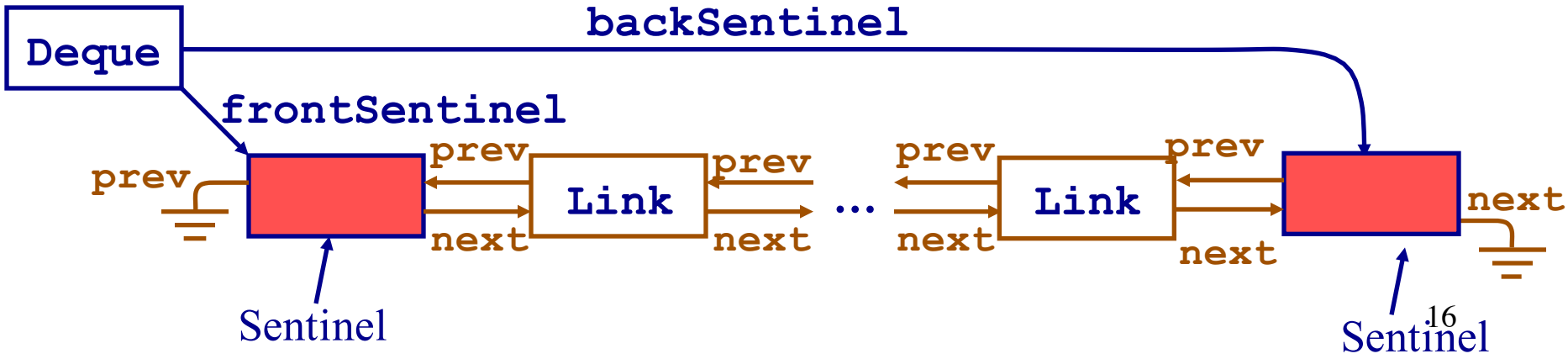
Deque

- Add/remove from both front and back
- Must maintain forward and backward links



If One Sentinel is Good

- Add a sentinel to BOTH front and back
- Eliminates the need to handle special cases
- Most "real" tools use two sentinels (e.g., Java)



Deque Interface

```
int isEmpty();
void addFront(TYPE val);   /*Add value at front*/
void addBack (TYPE val);   /*Add value at back*/
void removeFront();        /*Remove value at front*/
void removeBack ();        /*Remove value at back*/
TYPE front();              /*Get value at front*/
TYPE back();               /*Get value at back*/
```

Deque Structure

```
struct dlink {
    TYPE value;
    struct dlink * next;
    struct dlink * prev;
};

struct listDeque {
    int size;
    struct dlink * frontSentinel;
    struct dlink * backSentinel;
};
```

initDeque

```
void initDeque (struct listDeque *dq) {
   dq->frontSentinel = (struct dlink *)
                       malloc(sizeof(struct dlink));
   assert(dq->frontSentinel != 0);

   dq->backSentinel = (struct dlink *)
                      malloc(sizeof(struct dlink));
   assert(dq->backSentinel != 0);

     ...
}
```

initDeque

```
void initDeque (struct listDeque *dq) {
   dq->frontSentinel = (struct dlink *)
                       malloc(sizeof(struct dlink));
   assert(dq->frontSentinel != 0);
   dq->backSentinel = (struct dlink *)
                       malloc(sizeof(struct dlink));
   assert(dq->backSentinel != 0);
   dq->frontSentinel->next = dq->backSentinel;
   dq->backSentinel->prev = dq->frontSentinal;
   dq->size = 0;
}
```

Add to Front or Back

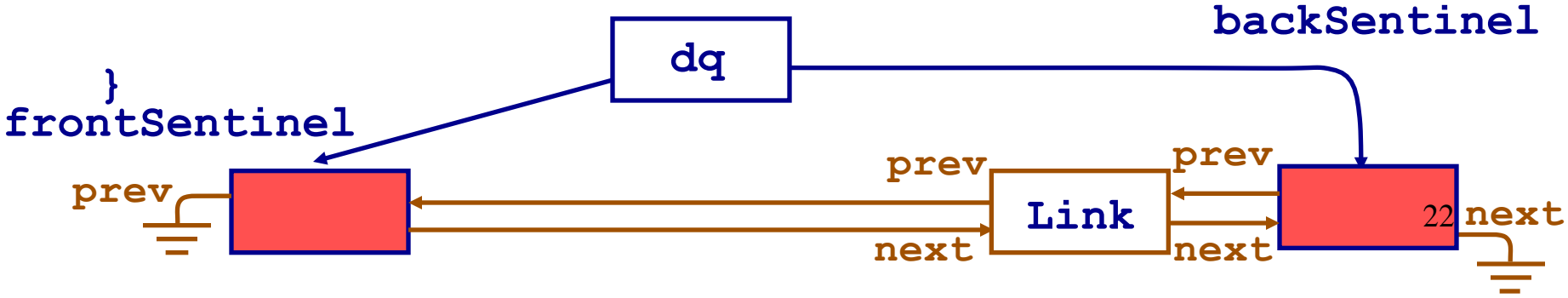
```
void addFrontDeque (struct listDeque *dq, TYPE e)
{
      _addDeque(dq, dq->frontSentinel->next, e);
}
```

```
void addBackDeque (struct listDeque *dq, TYPE e)
{
      _addDeque(dq, dq->backSentinel, e);
}
```

Add Double Link to Deque

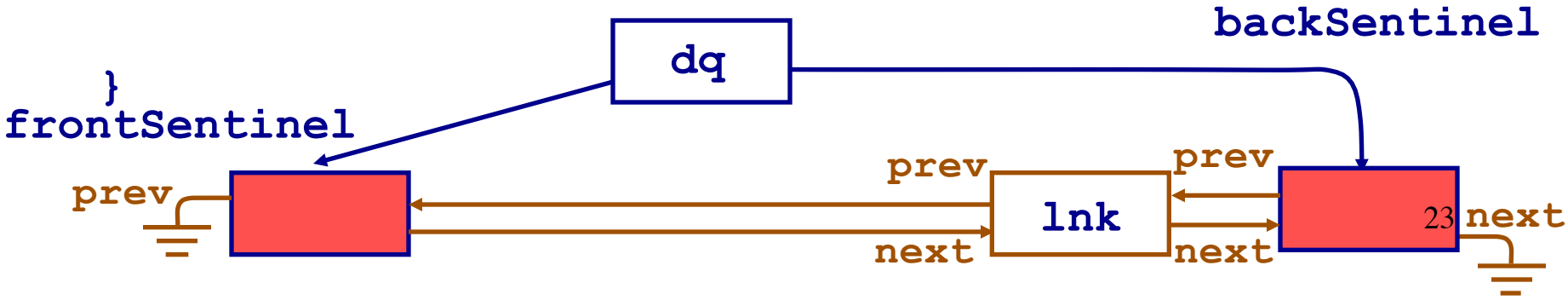
```
void _addDeque (struct listDeque *dq,
                struct dlink *lnk, TYPE e){

    ...
```

What is *lnk?



Add Double Link to Deque

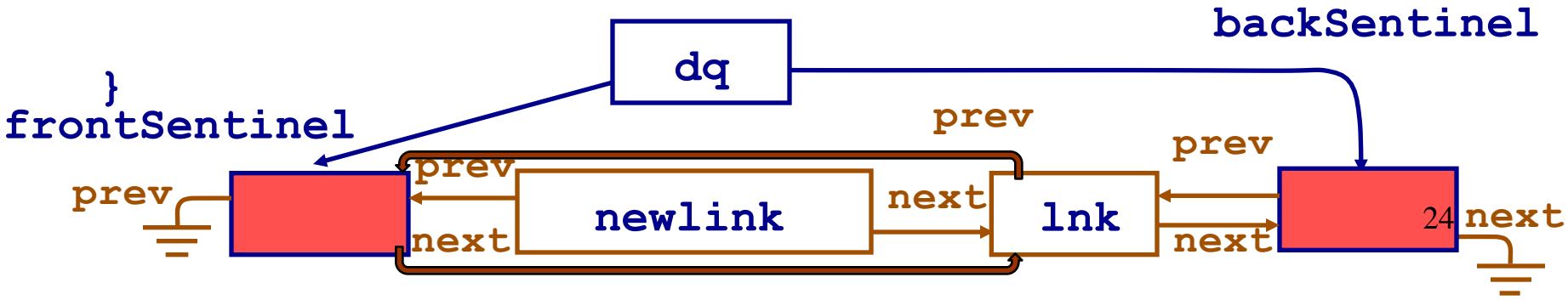
```
void _addDeque (struct listDeque *dq,
                struct dlink *lnk, TYPE e){
    struct dlink * newlink = (struct dlink *)
                    malloc(sizeof(struct dlink));
    assert(newlink != 0);
    newlink->value = e;

  ...
```



Add Double Link to Deque

```
void _addDeque (struct listDeque *dq,
                struct dlink *lnk, TYPE e){
    struct dlink * newlink = (struct dlink *)
                    malloc(sizeof(struct dlink));
    assert(newlink != 0);
    newlink->value = e;
    newlink->prev = lnk->prev;
    newlink->next = lnk;
```

If adding to front



Add Double Link to Deque

```
void _addDeque (struct listDeque *dq,
                struct dlink *lnk, TYPE e){
    struct dlink * newlink = (struct dlink *)
                    malloc(sizeof(struct dlink));
    assert(newlink != 0);
    newlink->value = e;
    newlink->prev = lnk->prev;
    newlink->next = lnk;
    lnk->prev->next = newlink;
    lnk->prev = newlink;
    dq->size++;
}
```