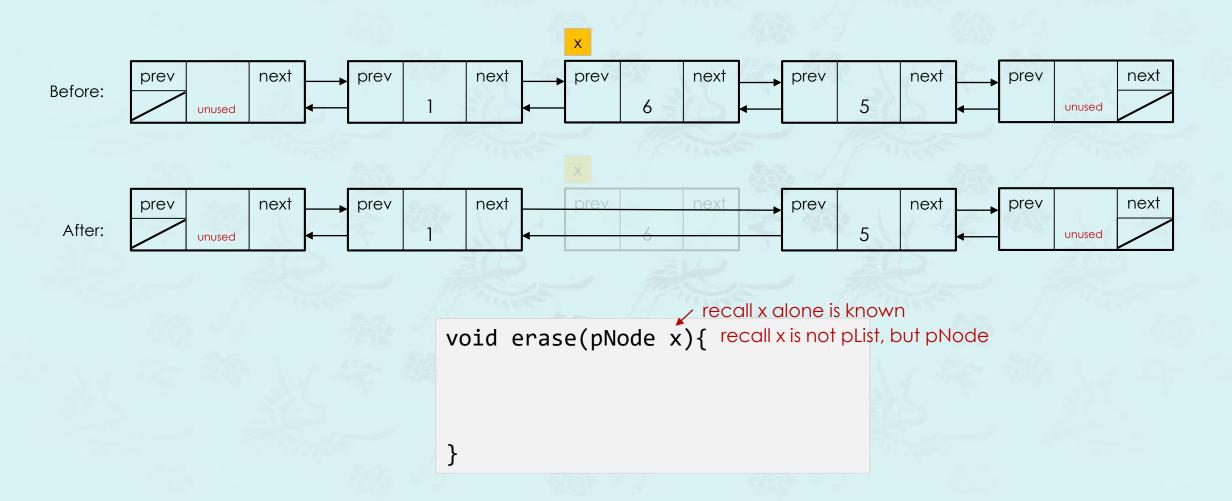
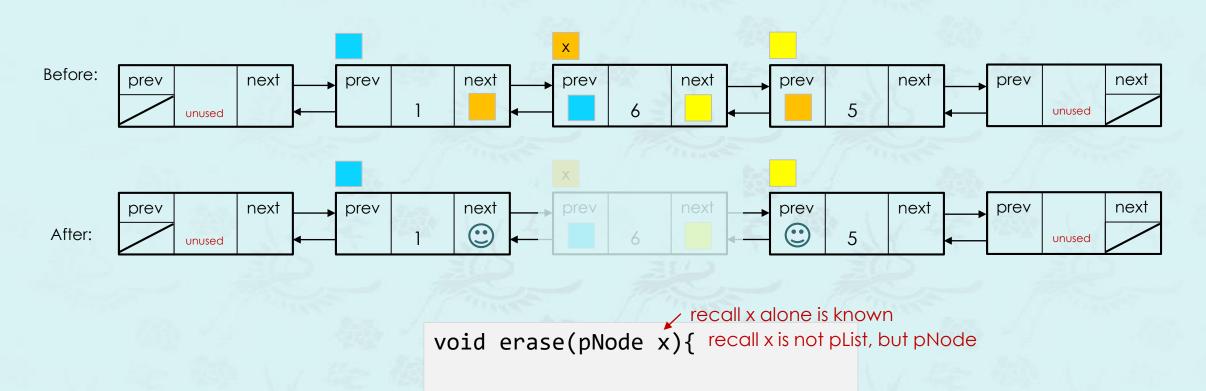
Data Structures Chapter 4

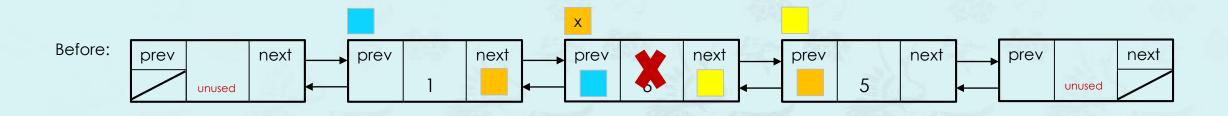
- 1. Singly Linked List
- 2. Doubly Linked List
 - Revisit Singly Linked List
 - Sentinel Nodes & Basic Operations
 - Two Key Operations: erase, insert
 - Advanced Operations





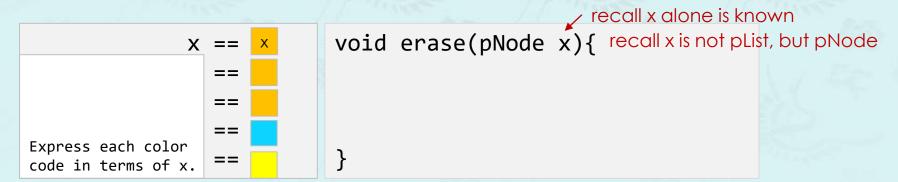


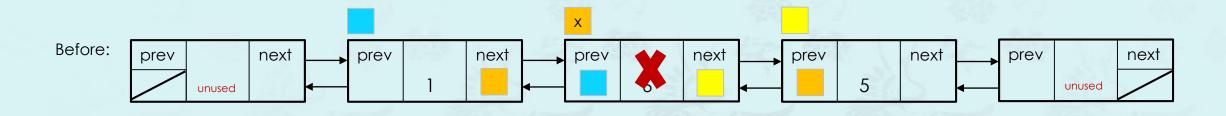




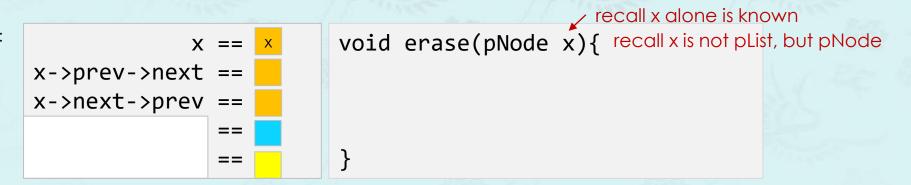
After:

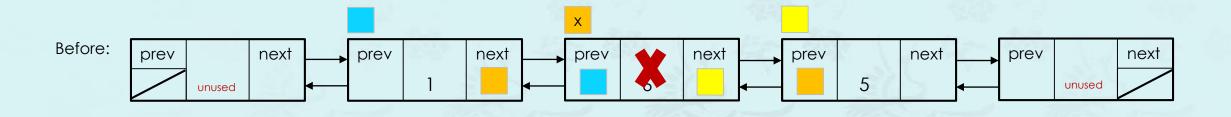






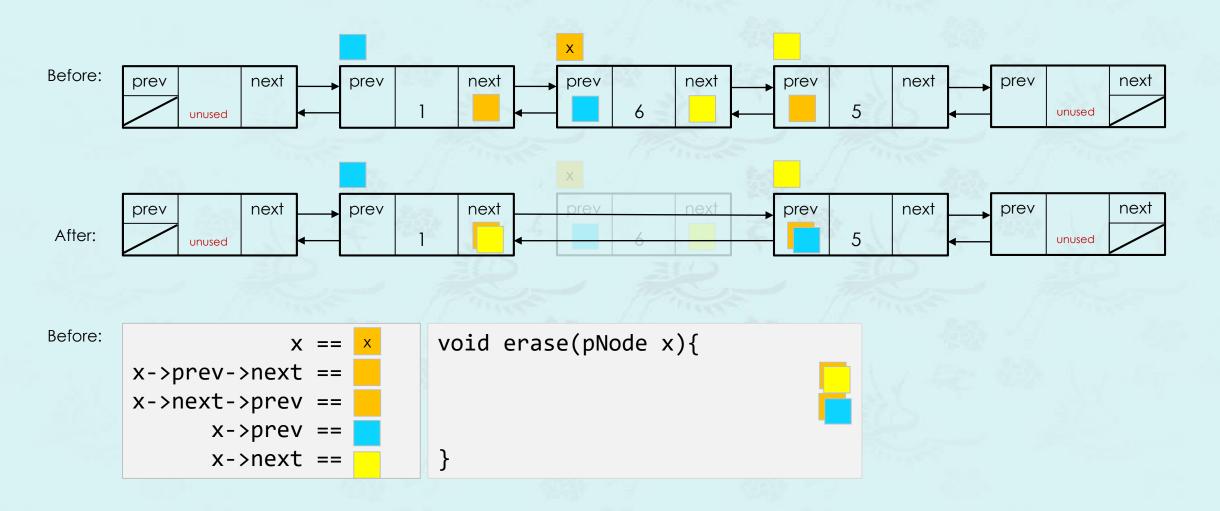
After:

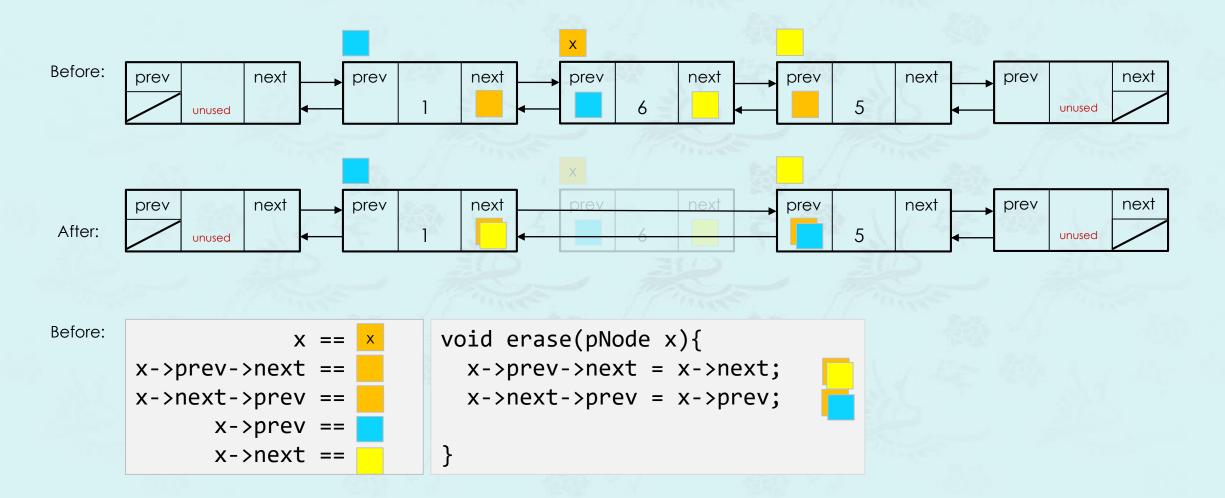


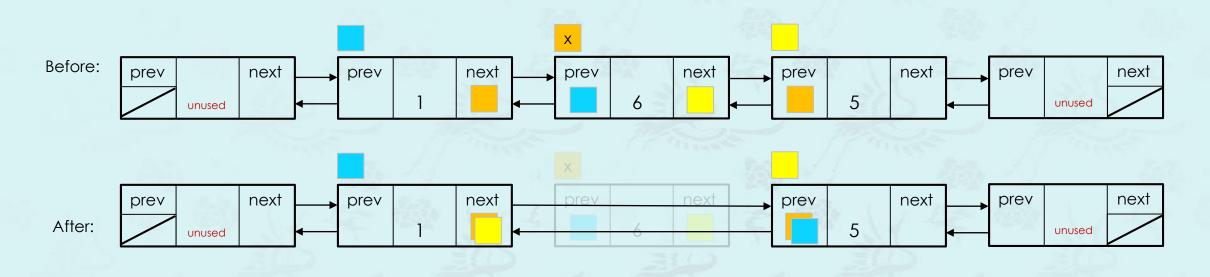


After:

```
x == x void erase(pNode x){ recall x alone is known
x == x void erase(pNode x) { recall x is not pList, but pNode
x->prev->next == x
x->next->prev == x
x->prev == x
x->next == }
```







```
x == x
void erase(pNode x){
x->prev->next == x->next;
x->next->prev == x->prev;
x->prev == delete x;
x->next == } //stay tuned for the better
```

Pop by value

Implement pop() using erase() and find().

```
void pop(pList p, int value){
  pNode node = find(p, value);
  erase(node);
}
```

```
void erase(pNode x){
  x->prev->next = x->next;
  x->next->prev = x->prev;
  delete x;
} //stay tuned for the better

pNode find(pList p, int value)
```

Pop by value

Implement pop() using erase() and find().

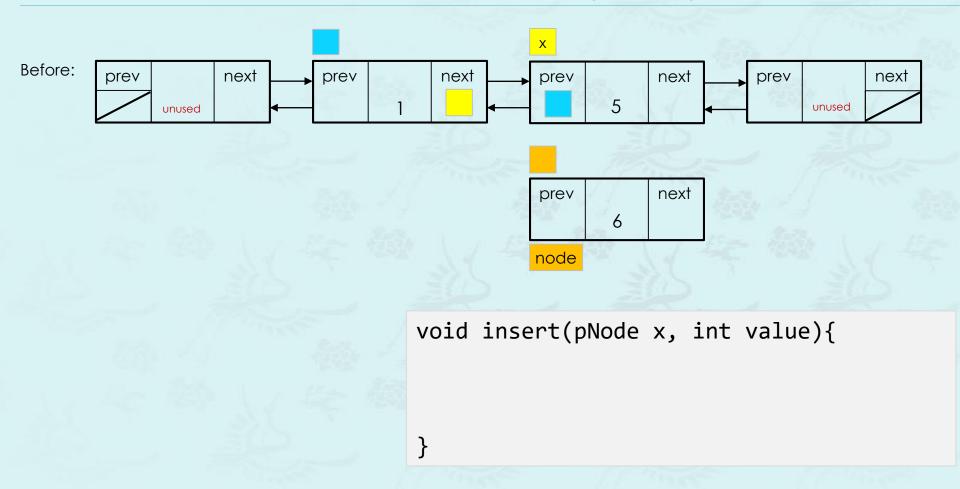
```
void pop(pList p, int value){
  pNode node = find(p, value);
  erase(node);
}
```

```
void pop(pList p, int value){
  erase(find(p, value));
} //stay tuned
```

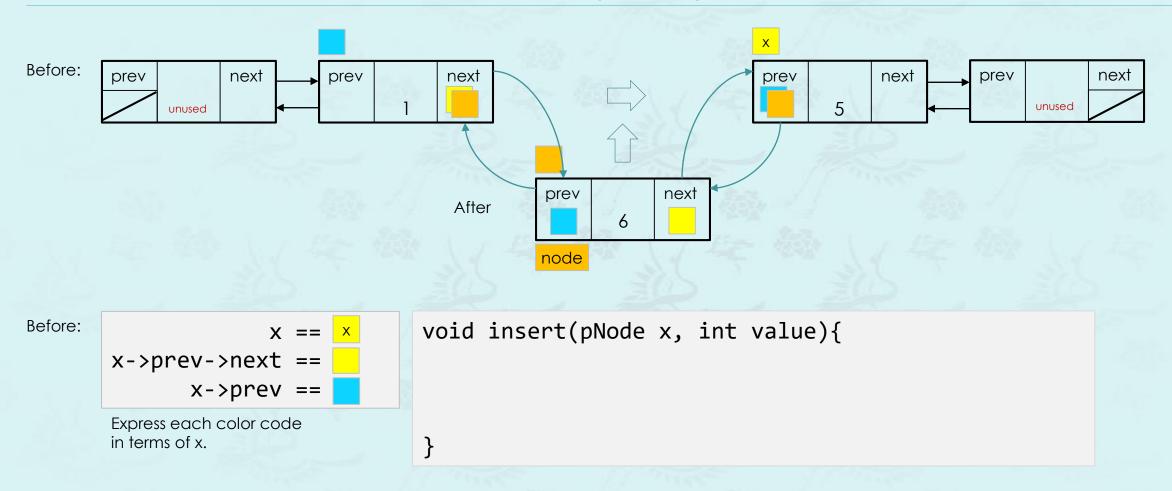
This code may not work some cases? How can you fix it?

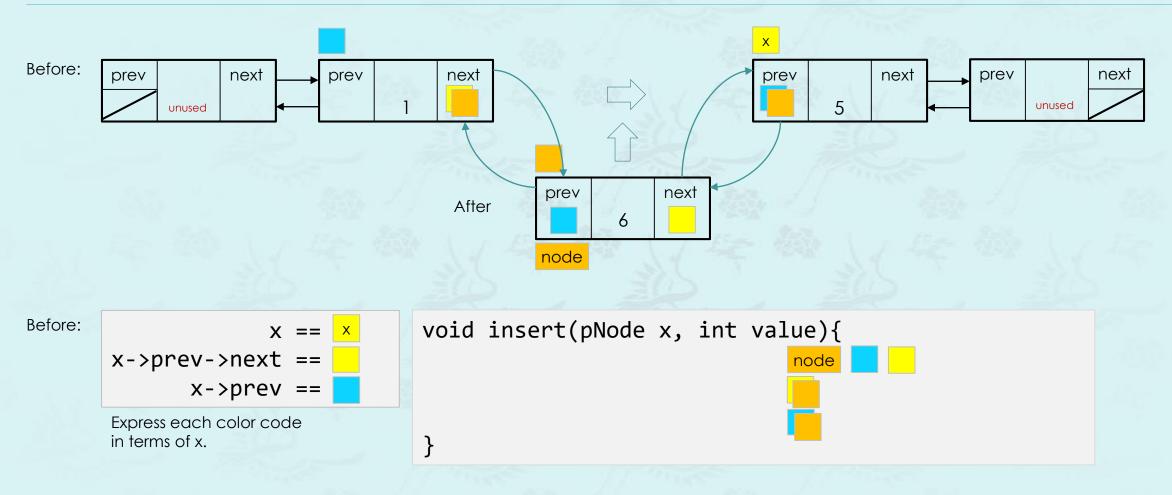
```
void erase(pNode x){
  x->prev->next = x->next;
  x->next->prev = x->prev;
  delete x;
} //stay tuned for the better

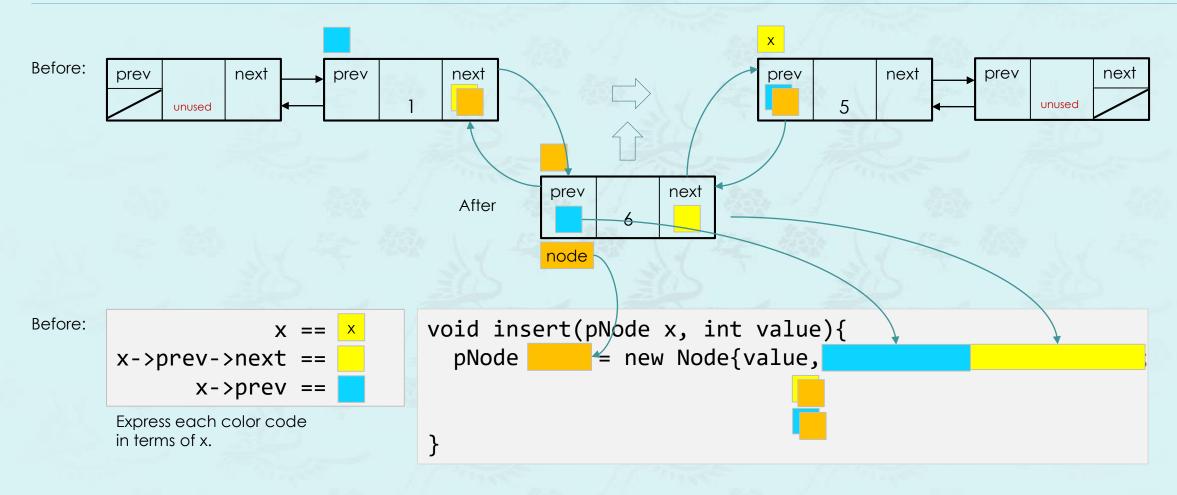
pNode find(pList p, int value)
```

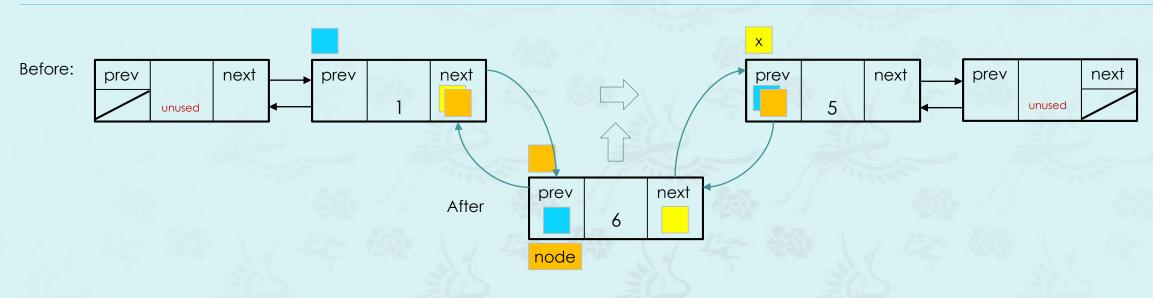






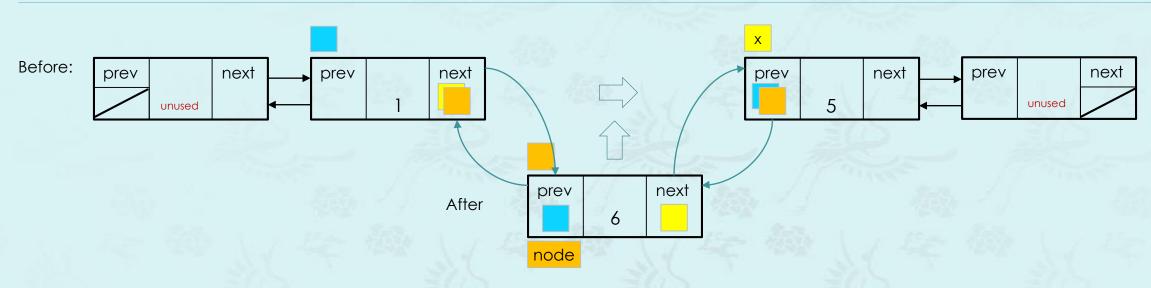






```
x == x
x->prev->next == x
x->prev == Express each color code in terms of x.
```

```
void insert(pNode x, int value){
  pNode node = new Node{value, x->prev, x};  x
  x->prev->next = node;
  x->prev = node;
}
```



Before:

Express each color code in terms of x.

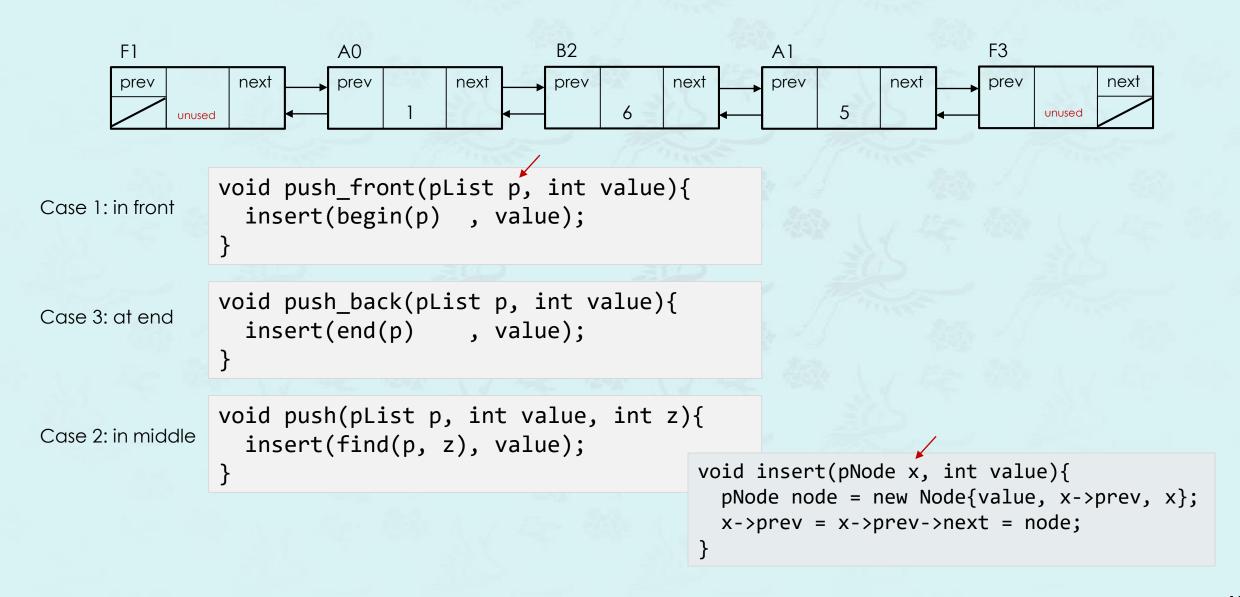
It is a matter of associativity!
Then, what is the associativity of '='?
right to left

```
void insert(pNode x, int value){
  pNode node = new Node{value, x->prev, x};  x
  x->prev->next = node;
  x->prev = node;
}
```

Can we replace two lines above with ____

- (1) x->prev = x->prev->next = node;
- (2) $x \rightarrow prev \rightarrow next = x \rightarrow prev = node;$
- (3) either one

push_front(), push_back(), push()



```
pNode begin(pList p);
                                       // returns the first node, not sentinel node
pNode end(pList p);
                                       // returns the ending sentinel node
                                      // returns the node in the middle of the list
pNode half(pList p)
pNode find(pList p, int value);
                                      // returns the first node with value
void clear(pList p);
                                       // free list of nodes
bool empty(pList p);
                                      // true if empty, false if no empty
int size(pList p);
                                      // returns size in the list
void insert(pNode x, int value);
                                       // inserts a new node with value at the node x
void erase (pNode x);
                                       // deletes a node and returns the previous node
                      stay tuned for enhancement
void push(pList p, int value, int z); // inserts a node with value at the node with x
void push front(pList p, int value);  // inserts a node at front of the list
void push_back(pList p, int value); // inserts a node with value at end of the list
void push sorted(pList p, int value, bool ascending = true); // inserts a node in sorted
void pop(pList p, int value);
                                       // deletes the first node with value
void pop front(pList p);
                                      // deletes the first node in the list
void pop back(pList p);
                                   // deletes the last node in the list, O(1)
                        // deletes all the nodes O(n)
void pop_backN(pList p);
void pop all(pList p, int value);  // deletes all the nodes with value
pList sort(pList p);
                                      // returns a `new list` sorted
bool sorted(pList p);
                                      // returns true if the list is sorted
void unique(pList p);
                                       // returns list with no duplicates, sorted
void reverse(pList p);
                                      // reverses the sequence
void shuffle(pList p);
                                       // shuffles the list
void show(pList p);
                                       // shows all data items in linked list
```

Summary & quaestio qo < 9 9??

Data Structures Chapter 4

- 1. Singly Linked List
- 2. Doubly Linked List
 - Revisit Singly Linked List
 - Sentinel Nodes & Basic Operations
 - Two Key Operations: erase, insert
 - Advanced Operations