CS 32 Week 3 Discussion 1I

UCLA CS

Srinath

Topics

- Linked List:
  - Singly-linked list
  - Dummy nodes and singly-linked list with dummy nodes.
  - Doubly-linked list with dummy nodes.
  - Circular doubly linked list with dummy nodes.

Singly Linked List: define

A sequential data structure.

Advantage:

Easy to insert and delete without knowing the length.

```
struct Node {
  int val;
  Node* next;
};

Node* head = nullptr;
```

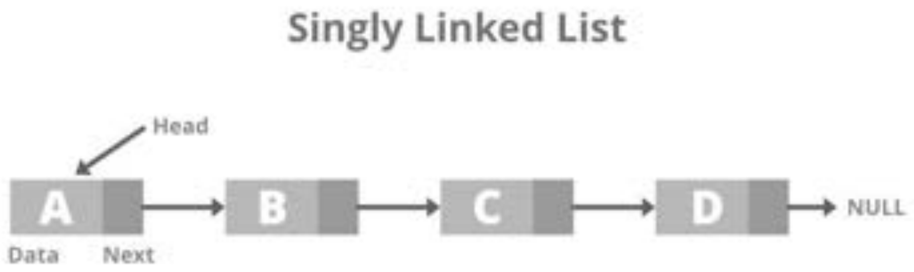


Image :https://www.geeksforgeeks.org

Singly Linked List: search by value

```
Node* Find_Val(Node* head, int x) {

}
```

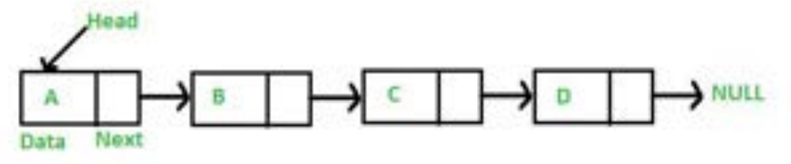
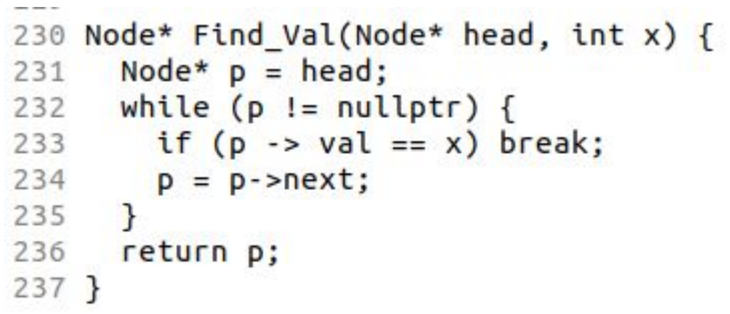


Image :https://www.geeksforgeeks.org

Singly Linked List: search by value



```
Node* Find_Val(Node* head, int x) {
  Node* p = head;
  while (p != nullptr) {
    if (p -> val == x) break;
    p = p->next;
  }
  return p;
}
```

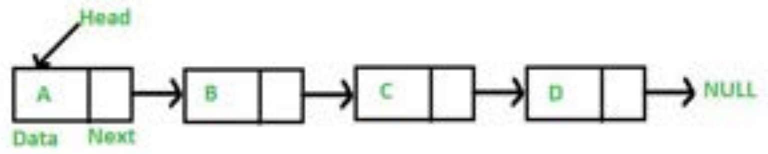


Image :https://www.geeksforgeeks.org

Singly Linked List: search by index

```
Node* Find_k_th(Node* head, int k) {

}
```

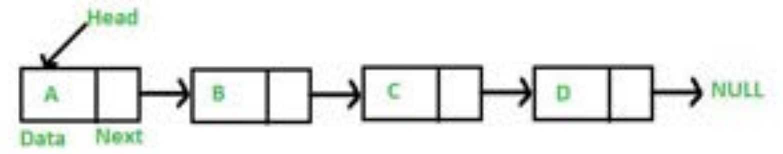
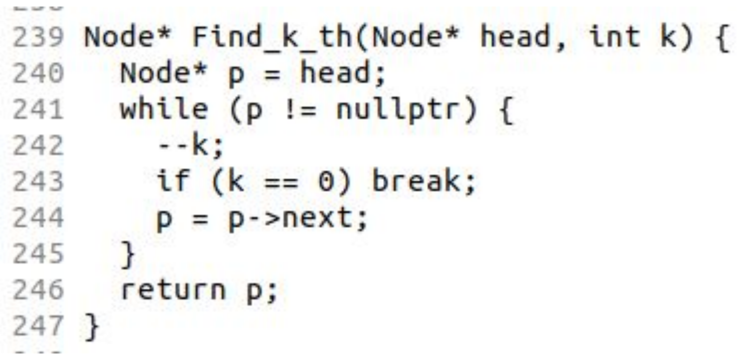


Image :https://www.geeksforgeeks.org

Singly Linked List: search by index



```
Node* Find_k_th(Node* head, int k) {
  Node* p = head;
  while (p != nullptr) {
    --k;
    if (k == 0) break;
    p = p->next;
  }
  return p;
}
```

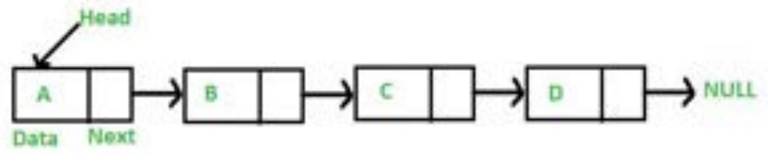
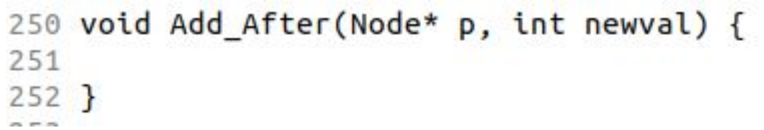


Image :https://www.geeksforgeeks.org

Singly Linked List: insert an element after p



```
void Add_After(Node* p, int newval) {

}
```

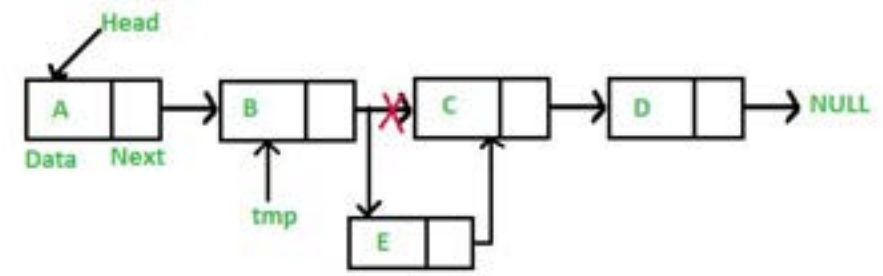


Image :https://www.geeksforgeeks.org

Singly Linked List: insert an element after p

```
void Add_After(Node* p, int x) {
  Node* newnode = new Node;
  newnode->val = x;
  if (p == head && head == nullptr) {
    newnode->next = nullptr;
    head = newnode;
  }
  else {
    newnode->next = p->next;
    p->next = newnode;
  }
}
```

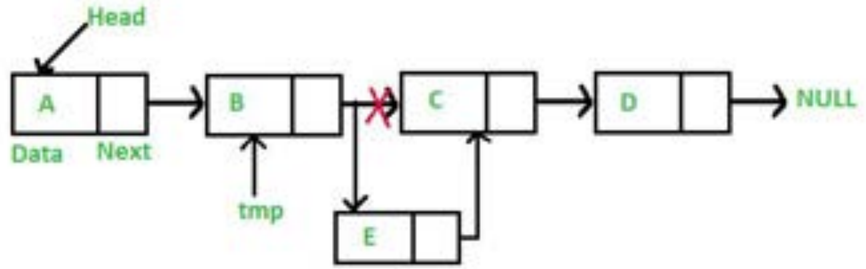
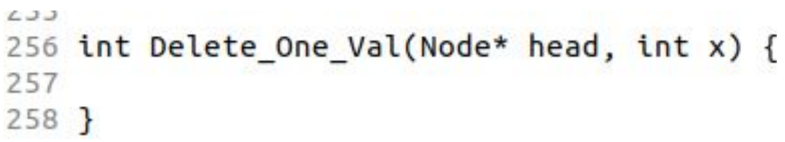


Image :https://www.geeksforgeeks.org

Singly Linked List: delete first node with value x



```
int Delete_One_Val(Node* head, int x) {

}
```

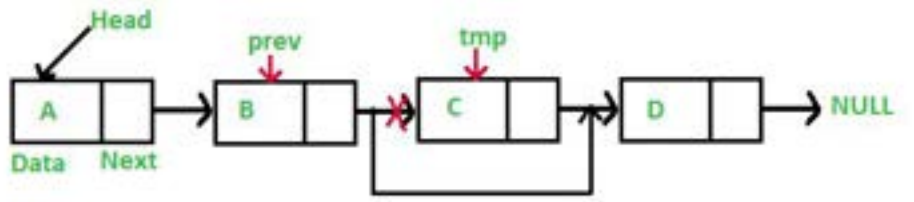
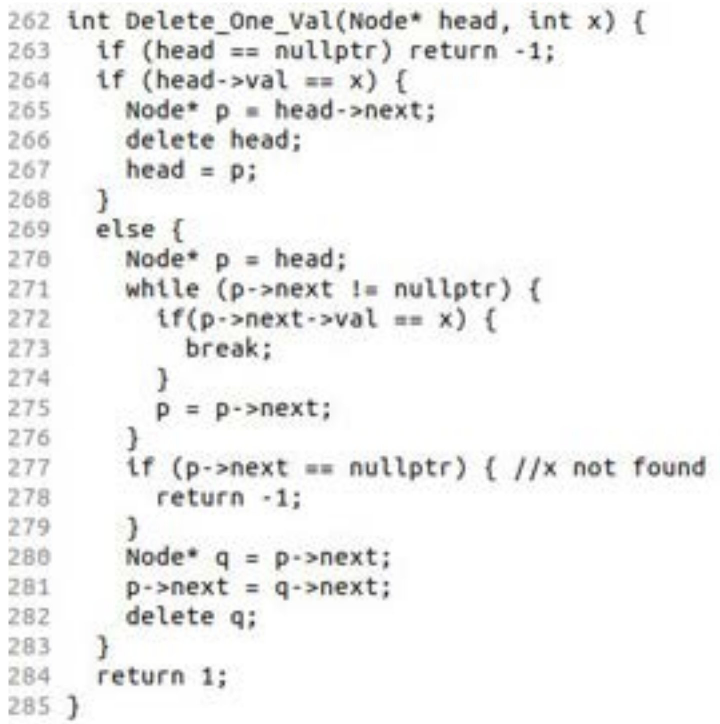


Image :https://www.geeksforgeeks.org

Singly Linked List: delete first node with value x



```
int Delete_One_Val(Node* head, int x) {
  if (head == nullptr) return -1;
  if (head->val == x) {
    Node* p = head->next;
    delete head;
    head = p;
  }
  else {
    Node* p = head;
    while (p->next != nullptr) {
      if(p->next->val == x) {
        break;
      }
      p = p->next;
    }
    if (p->next == nullptr) { //x not found
      return -1;
    }
    Node* q = p->next;
    p->next = q->next;
    delete q;
  }
  return 1;
}
```

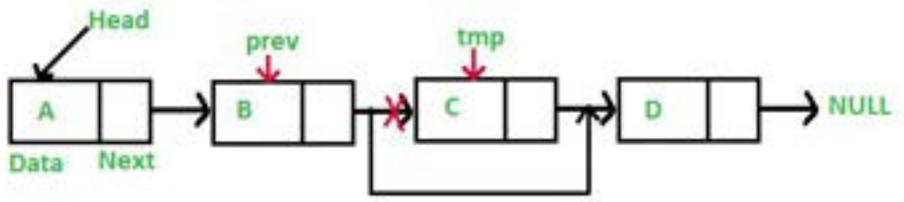


Image :https://www.geeksforgeeks.org

Singly Linked List: reverse the order of a linked list

```
void Reverse_Order(Node* head) {

}
```

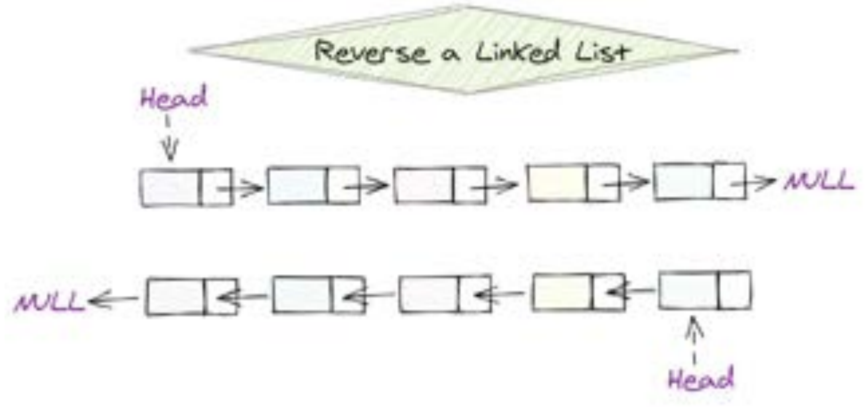
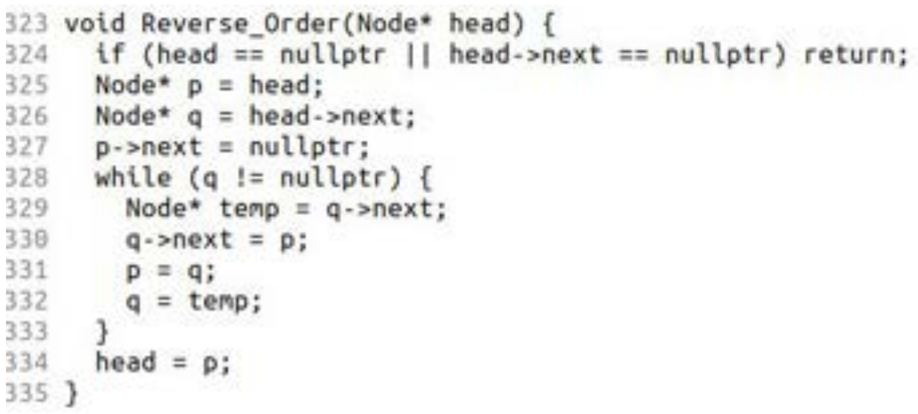


Image: https://algodaily.com/challenges/reverse-a-linked-list

Singly Linked List: reverse the order of a linked list



```
void Reverse_Order(Node* head) {
  if (head == nullptr || head->next == nullptr) return;
  Node* p = head;
  Node* q = head->next;
  p->next = nullptr;
  while (q != nullptr) {
    Node* temp = q->next;
    q->next = p;
    p = q;
    q = temp;
  }
  head = p;
}
```

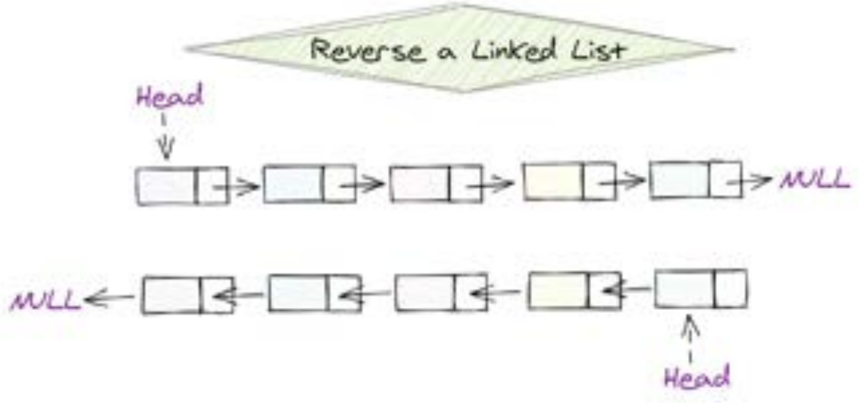
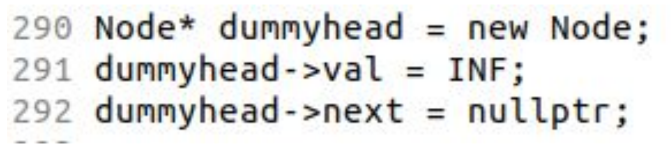


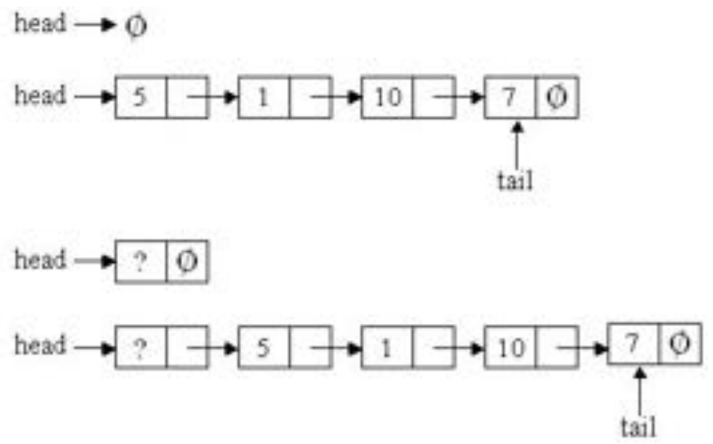
Image: https://algodaily.com/challenges/reverse-a-linked-list

Singly Linked List: dummy node for head

Sometimes, adding a dummy node simplifies the code. The first element becomes head->next.



```
Node* dummyhead = new Node;
dummyhead->val = INF;
dummyhead->next = nullptr;
```



Singly Linked List: insert a node after p (dummy node)

```
void Add_After(Node* p, int newval) {
  Node* newnode = new Node;
  newnode->val = newval;
  newnode->next = p->next;
  p->next = newnode;
}
```

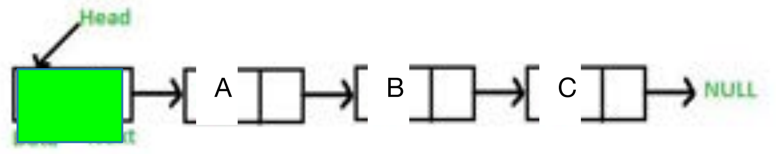


Image :https://www.geeksforgeeks.org

Singly Linked List: delete first node with value x (dummy node)

```
int Delete_One_Val(Node* head, int x) {
  Node* p = head;
  while (p->next != nullptr) {
    if (p->next->val == x)
      break;
    p = p->next;
  }
  if (p->next == nullptr) return -1;
  Node* q = p->next;
  p->next = q->next;
  delete q;
  return 1;
}
```

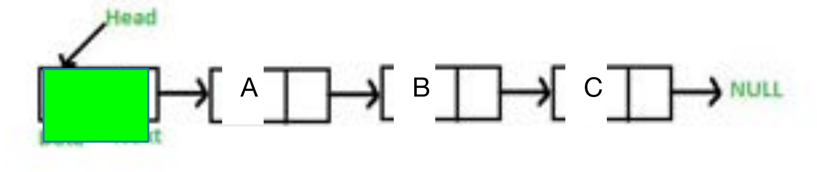
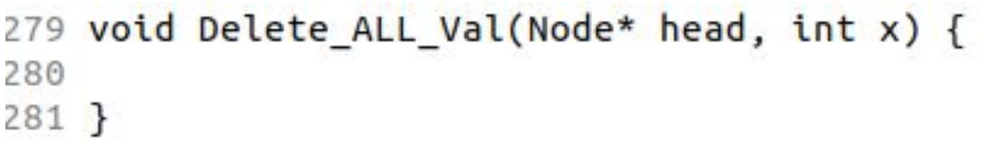


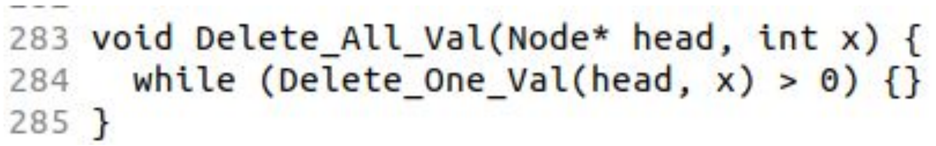
Image :https://www.geeksforgeeks.org

Singly Linked List: delete all nodes with value x



```
void Delete_ALL_Val(Node* head, int x) {

}
```

Singly Linked List: delete all nodes with value x



```
void Delete_All_Val(Node* head, int x) {
  while (Delete_One_Val(head, x) > 0) {}
}
```

Doubly Linked List

```
struct DNode {
  int val;
  DNode* prev, next;
};

DNode* head = nullptr;
DNode* tail = nullptr;
```

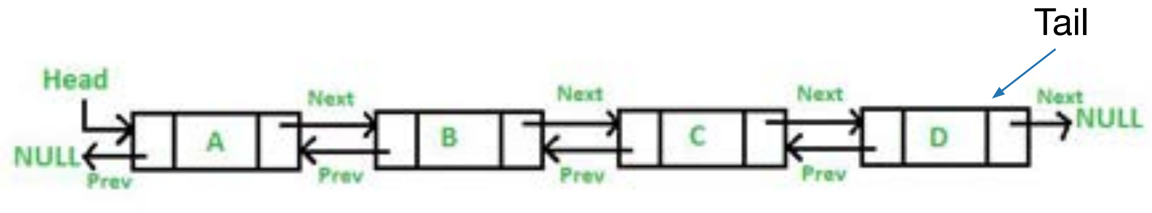


Image :https://www.geeksforgeeks.org

Doubly Linked List (dummy nodes)

```
struct DNode {
  int val;
  DNode* prev, next;
};

DNode* dummyhead = new DNode;
dummyhead->val = INF;
dummyhead->next = dummytail;
dummyhead->prev = nullptr;
DNode* dummytail = new DNode;
dummytail->val = INF;
dummytail->next = nullptr;
dummytail->prev = dummyhead;
```

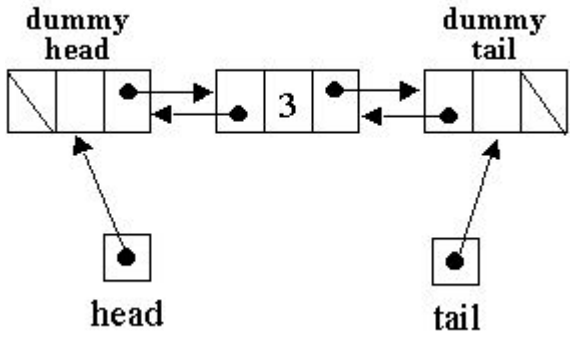
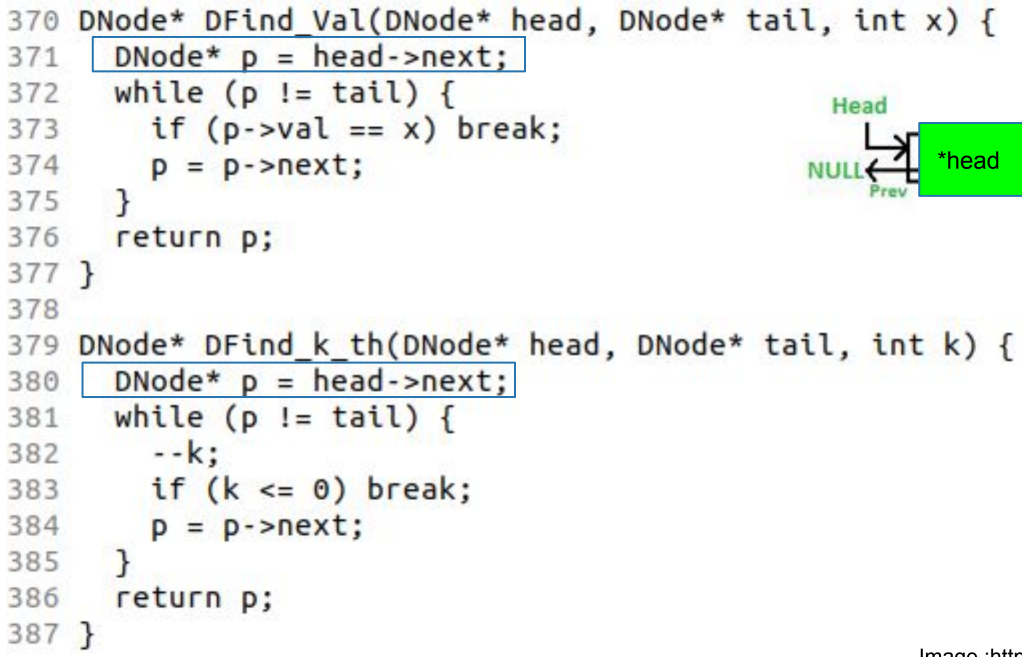


Image: https://condor.depaul.edu/ntomuro/courses/300/notes/lecture10.html

Doubly Linked List: search by value or index

Searching by kth number or value are almost
the same as for the singly linked list.



```
DNode* DFind_Val(DNode* head, DNode* tail, int x) {
  DNode* p = head->next;
  while (p != tail) {
    if (p->val == x) break;
    p = p->next;
  }
  return p;
}

DNode* DFind_k_th(DNode* head, DNode* tail, int k) {
  DNode* p = head->next;
  while (p != tail) {
    --k;
    if (k <= 0) break;
    p = p->next;
  }
  return p;
}
```

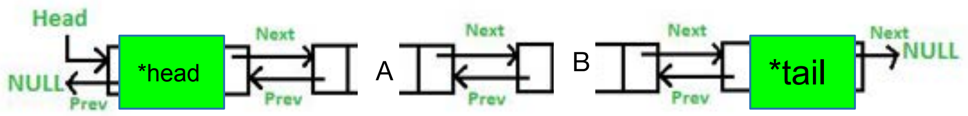
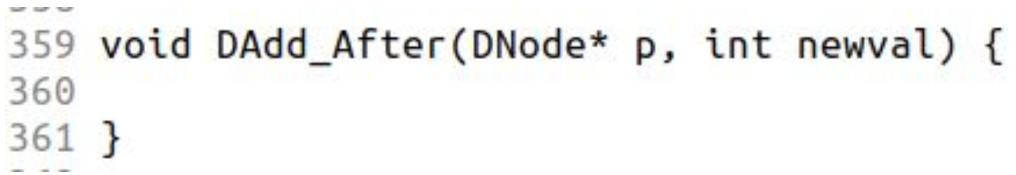


Image :https://www.geeksforgeeks.org

Doubly Linked List: insert a new node after p



```
void DAdd_After(DNode* p, int newval) {

}
```

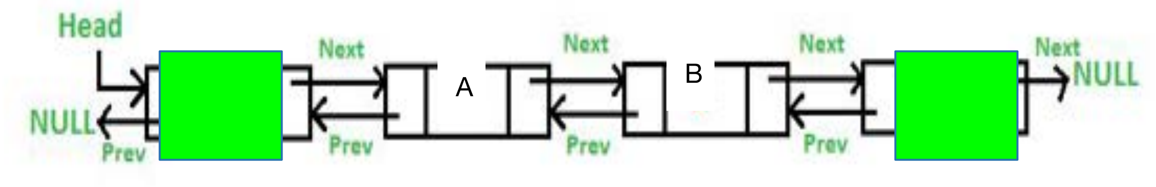
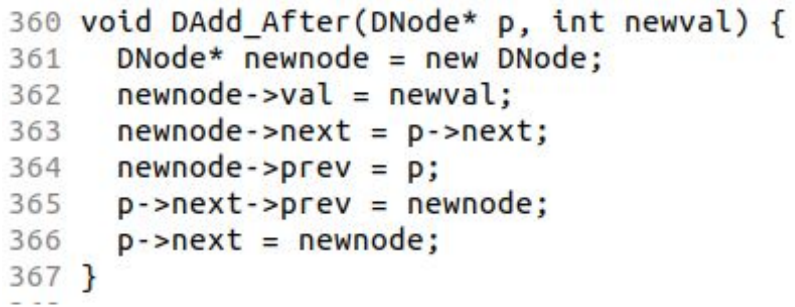


Image :https://www.geeksforgeeks.org

Doubly Linked List: insert a new node after p



```
void DAdd_After(DNode* p, int newval) {
  DNode* newnode = new DNode;
  newnode->val = newval;
  newnode->next = p->next;
  newnode->prev = p;
  p->next->prev = newnode;
  p->next = newnode;
}
```

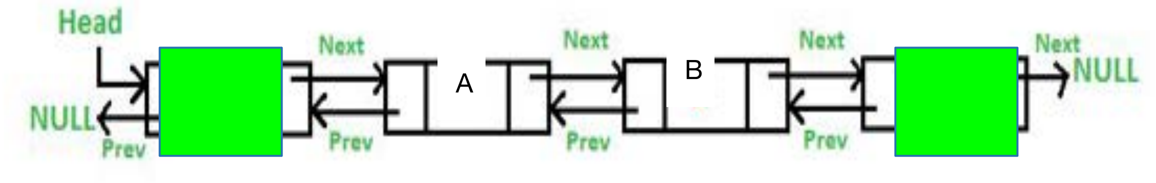


Image :https://www.geeksforgeeks.org

Doubly Linked List: delete a DNode* p

```
DNode* DDelete_One_Node(DNode* p) {

}
```

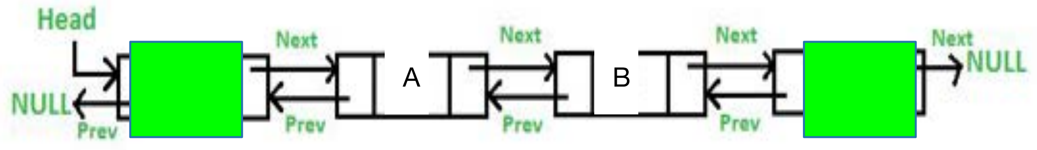
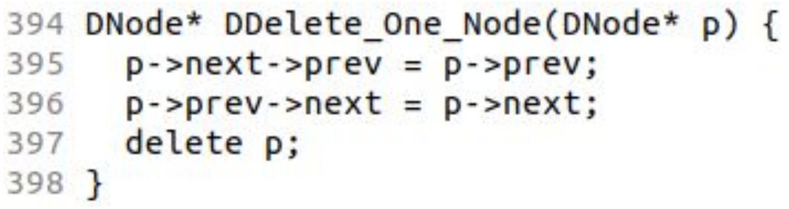
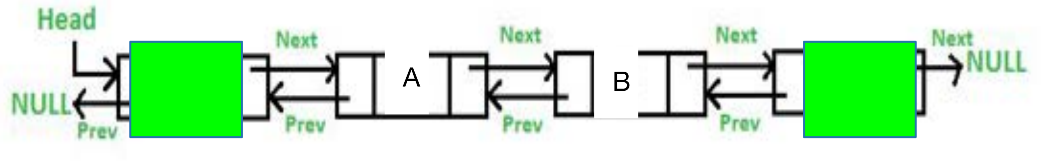


Image :https://www.geeksforgeeks.org

Doubly Linked List: delete a DNode* p



```
DNode* DDelete_One_Node(DNode* p) {
  p->next->prev = p->prev;
  p->prev->next = p->next;
  delete p;
}
```

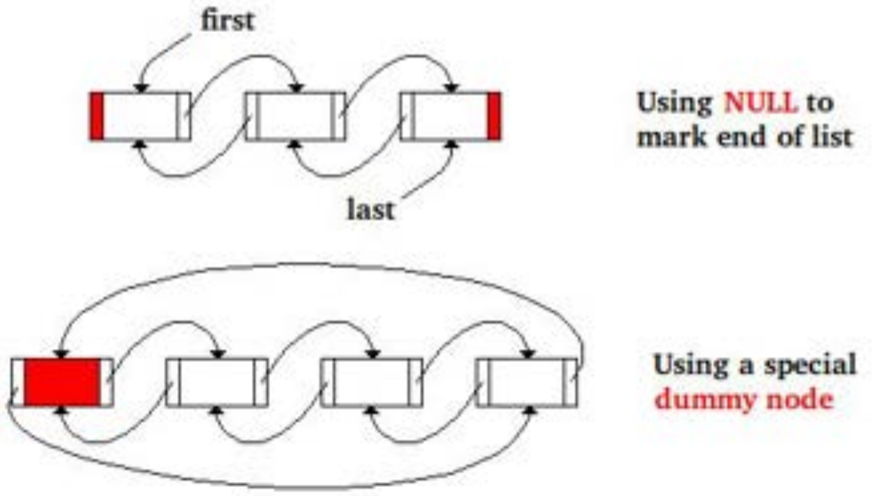


Circular Doubly Linked List

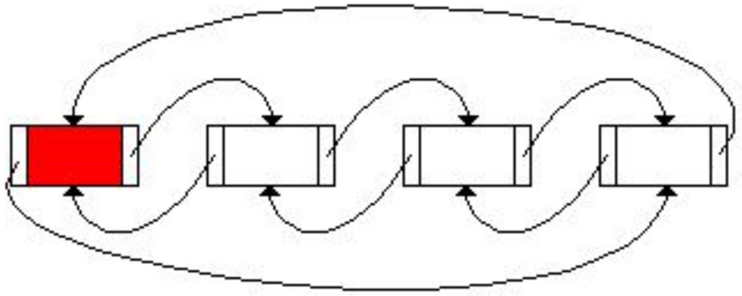
Instead of having a head and a tail, why not connect the head and tail to make it circular?

Now only one dummy head is needed.

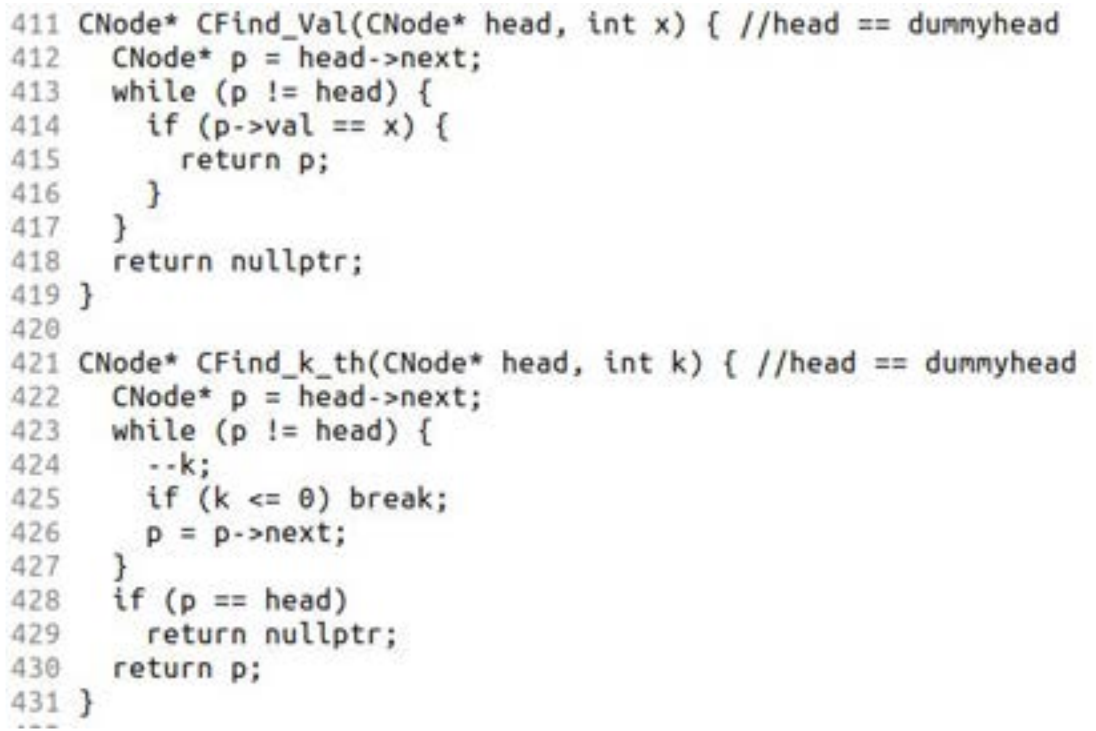
```
struct CNode {
  int val;
  CNode* prev, next;
};

CNode* dummyhead = new CNode;
dummyhead->val = INF;
dummyhead->prev = dummyhead;
dummyhead->next = dummyhead;
```



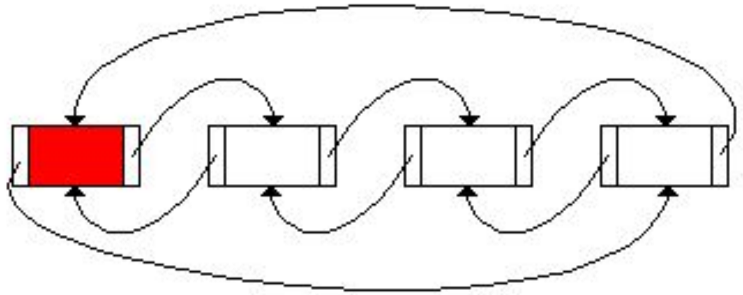
Circular Doubly Linked List: search by value or index



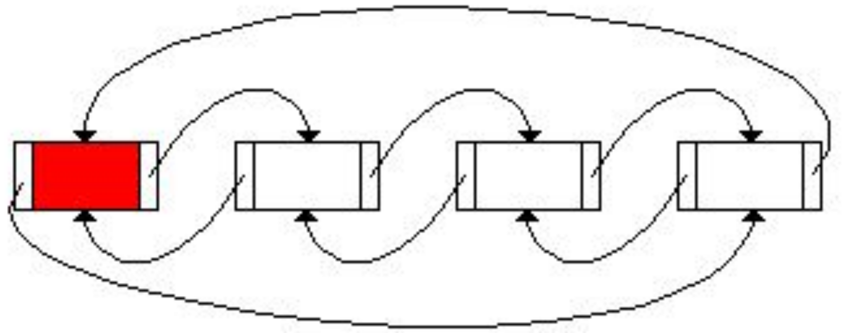
Circular Doubly Linked List: search by value or index



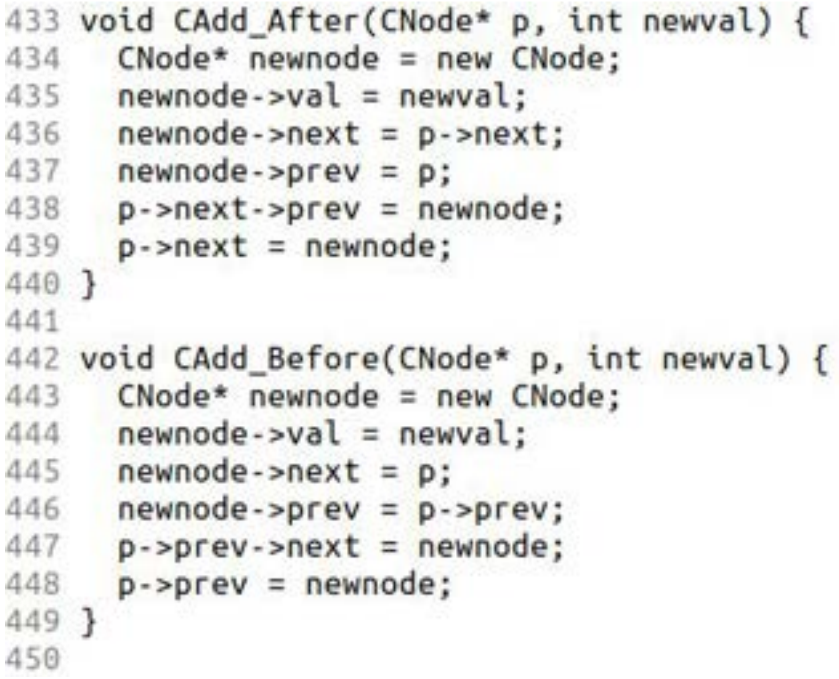
```
CNode* CFind_Val(CNode* head, int x) { //head == dummyhead
  CNode* p = head->next;
  while (p != head) {
    if (p->val == x) {
      return p;
    }
  }
  return nullptr;
}

CNode* CFind_k_th(CNode* head, int k) { //head == dummyhead
  CNode* p = head->next;
  while (p != head) {
    --k;
    if (k <= 0) break;
    p = p->next;
  }
  if (p == head)
    return nullptr;
  return p;
}
```



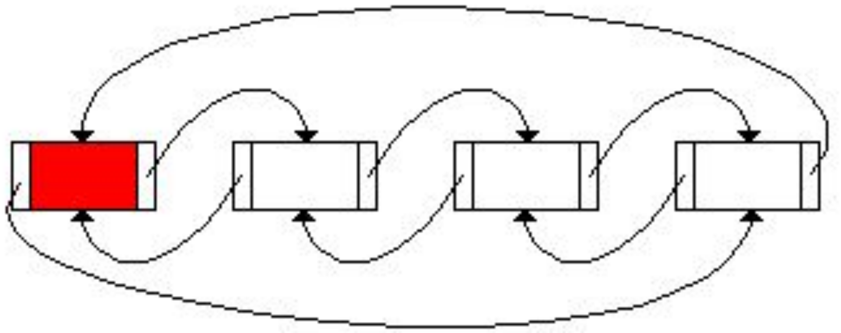
Circular Doubly Linked List: insert after/before Node p



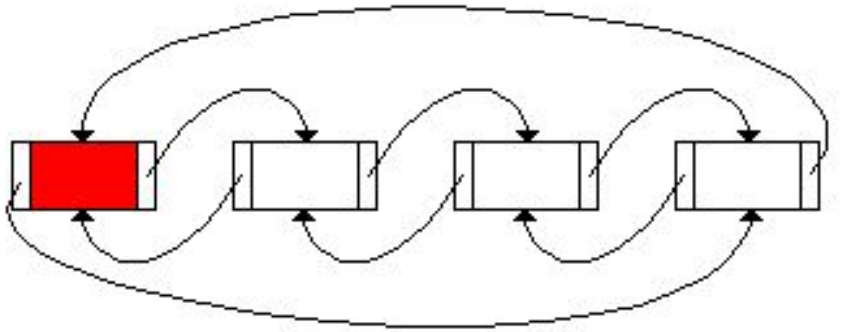
Circular Doubly Linked List: insert after/before Node p



```
void CAdd_After(CNode* p, int newval) {
  CNode* newnode = new CNode;
  newnode->val = newval;
  newnode->next = p->next;
  newnode->prev = p;
  p->next->prev = newnode;
  p->next = newnode;
}

void CAdd_Before(CNode* p, int newval) {
  CNode* newnode = new CNode;
  newnode->val = newval;
  newnode->next = p;
  newnode->prev = p->prev;
  p->prev->next = newnode;
  p->prev = newnode;
}

```

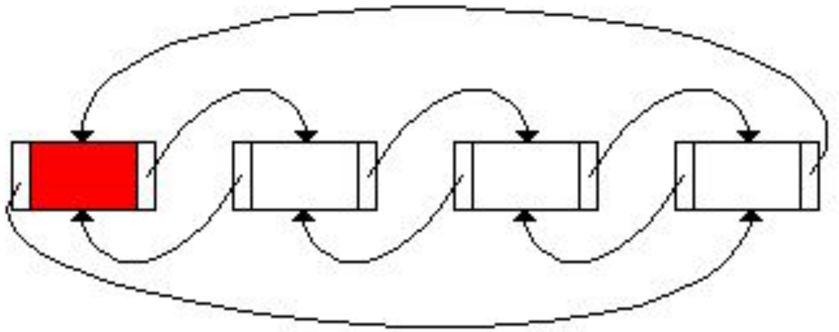


Circular Doubly Linked List: delete Node p



Circular Doubly Linked List: delete Node p

```
void CDelete_One_Node(CNode* p) {
  //assume p != dummyhead
  p->next->prev = p->prev;
  p->prev->next = p->next;
  delete p;
}
```



Linked Lists

1. Singly-linked list.
   a. For simple tasks.
   b. Keeping next only.
2. Doubly-linked list.
   a. Bidirectional, making some tasks like deletion and reverse order easier.
   b. Keeping prev and next.
3. Circular doubly linked list (my personal favorite).
   a. Bidirectional and easy to implement.
   b. Keeping prev and next.

Time complexity: with n nodes in the list, O(n) complexity for insertion and deletion.

Question: how to swap two linked lists?

Exercise

Modify the insertion, Add_After(),to make the linked list sorted by value.