Data Structures & Algorithms

Prepared by: Mohamed Ayman

Machine Learning Researcher

spring 2019

sw.eng.MohamedAyman@gmail.com

facebook.com/sw.eng.MohamedAyman

linkedin.com/in/eng-mohamed-ayman

codeforces.com/profile/Mohamed_Ayman

Linked List

© Prepared by: Mohamed Ayman

Agenda

1- Linked List Definition

2- Linked List Representation

3- Types of Linked List

4- Basic Operations

5- Insertion Operation

6- Deletion Operation

7- Traversal Operation

8- Search Operation

Let's
STARTUP

Agenda

1- Linked List Definition

2- Linked List Representation

3- Types of Linked List

4- Basic Operations

5- Insertion Operation

6- Deletion Operation

7- Traversal Operation

8- Search Operation

Linked List Definition

- A linked list is a way to store a collection of elements. Like an array these can be character or integers. Each element in a linked list is stored in the form of a node.

- A linked list is a data structure in which the objects are arranged in a linear order. Unlike an array, however, in which the linear order is determined by the array indices, the order in a linked list is determined by a pointer in each object.

- Linked lists can be thought of from a high level perspective as being a series of nodes. Each node has at least a single pointer to the next node, and in the last node's case a null pointer representing that there are no more nodes in the linked list.

Agenda

✓ 1- Linked List Definition

2- Linked List Representation

3- Types of Linked List

4- Basic Operations

5- Insertion Operation

6- Deletion Operation

7- Traversal Operation

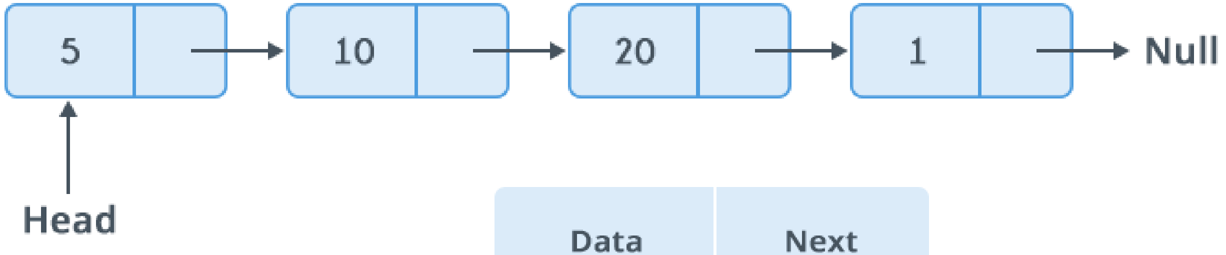
8- Search Operation

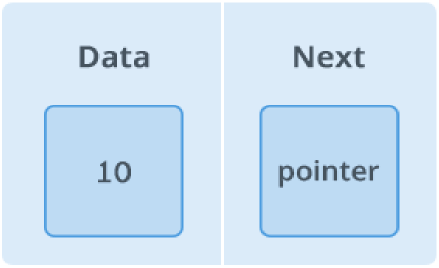
Linked List Representation

- Following are the important terms to understand the concept of Linked List:

Link: Each link of a linked list can store a data called an element.

Next: Each link of a linked list contains a link to the next link called Next.

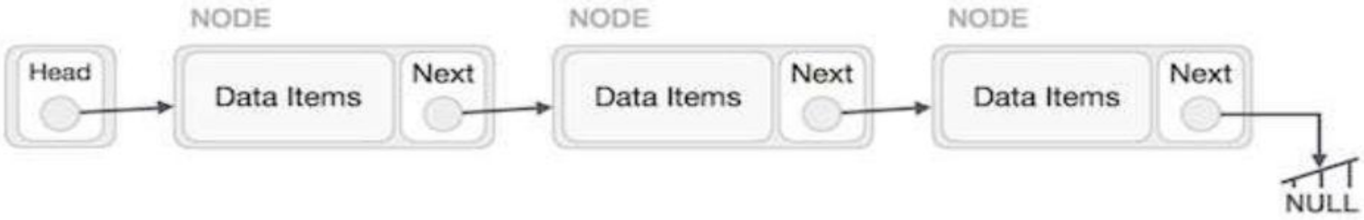




© Prepared by: Mohamed Ayman

Linked List Representation

Linked list can be visualized as a chain of nodes, where every node points to the next node.



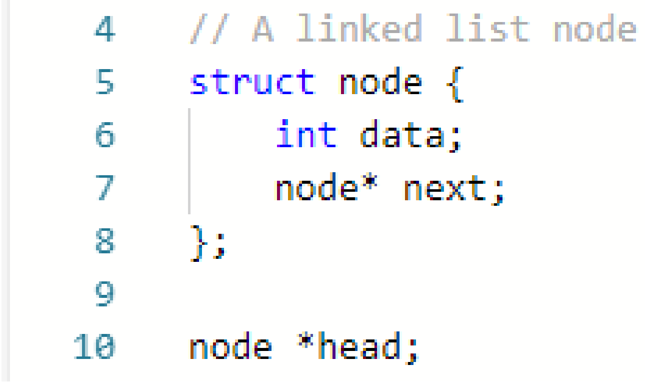
- Each link carries a data field(s) and a link field called next.

- Each link is linked with its next link using its next link.

- Last link carries a link as null to mark the end of the list.

Single Linked List Node in C++

Link: repl.it/repls/SmartSecondaryLanservers



```cpp
// A linked list node
struct node {
    int data;
    node* next;
};

node *head;
```

Agenda

✓ 1- Linked List Definition

✓ 2- Linked List Representation

3- Types of Linked List

4- Basic Operations

5- Insertion Operation

6- Deletion Operation

7- Traversal Operation

8- Search Operation

Types of Linked List

- Following are the various types of linked list:

1- Single Linked List:

Item navigation is forward only.

2- Doubly Linked List:

Items can be navigated forward and backward.

3- Circular Linked List:

Last item contains link of the first element as next and

the first element has a link to the last element as previous.

Agenda

- ✓ 1- Linked List Definition
- ✓ 2- Linked List Representation
- ✓ 3- Types of Linked List
- **4- Basic Operations**
- 5- Insertion Operation
- 6- Deletion Operation
- 7- Traversal Operation
- 8- Search Operation

Basic Operations

- Following are the basic operations supported by a list.

1- **Insertion:** Adds an element at the beginning or after beginning of the list.

2- **Deletion:** Deletes an element at the beginning or after beginning of the list.

3- **Traversal:** Displays the complete nodes in list.

4- **Search:** Searches an element using the given key.

Agenda

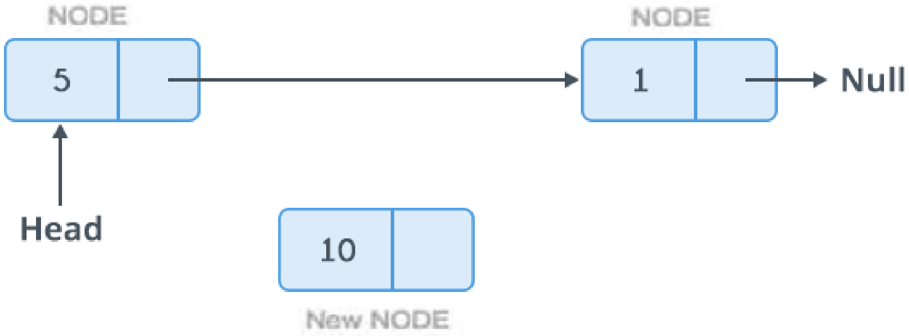
- ✓ 1- Linked List Definition
- ✓ 2- Linked List Representation
- ✓ 3- Types of Linked List
- ✓ 4- Basic Operations
- **5- Insertion Operation**
- 6- Deletion Operation
- 7- Traversal Operation
- 8- Search Operation

Insertion Operation

- Adding a new node in linked list is a more than one step activity.

We shall learn this with diagrams here.

1- First, create a node using the same structure and find the location where it has to be inserted.

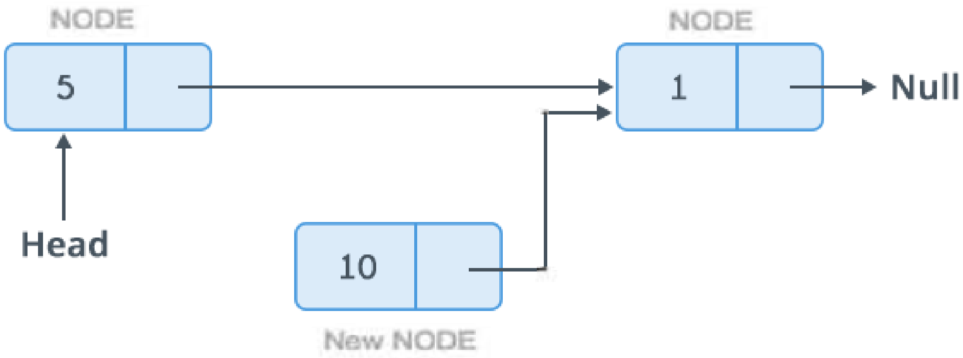


Insertion Operation

- Imagine that we are inserting a node B (NewNode), between A (LeftNode) and C (RightNode).

- Then point B.next to C : NewNode.next –› RightNode

It should look like this

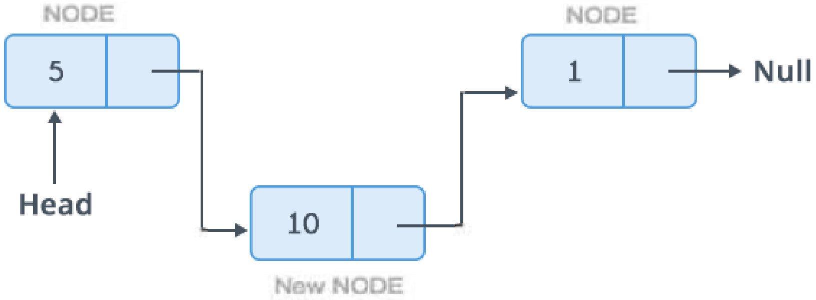


© Prepared by: Mohamed Ayman

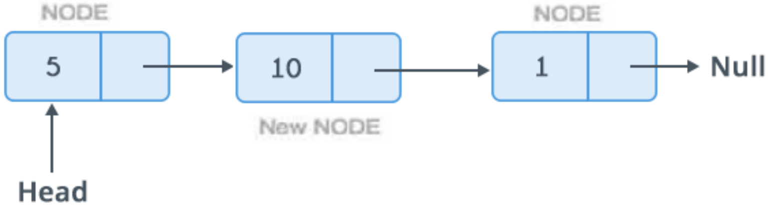
Insertion Operation

- Now, the next node at the left (A) should point to the new node (B) :

LeftNode.next —› NewNode



- This will put the new node in the middle of the two. The new list should look like this

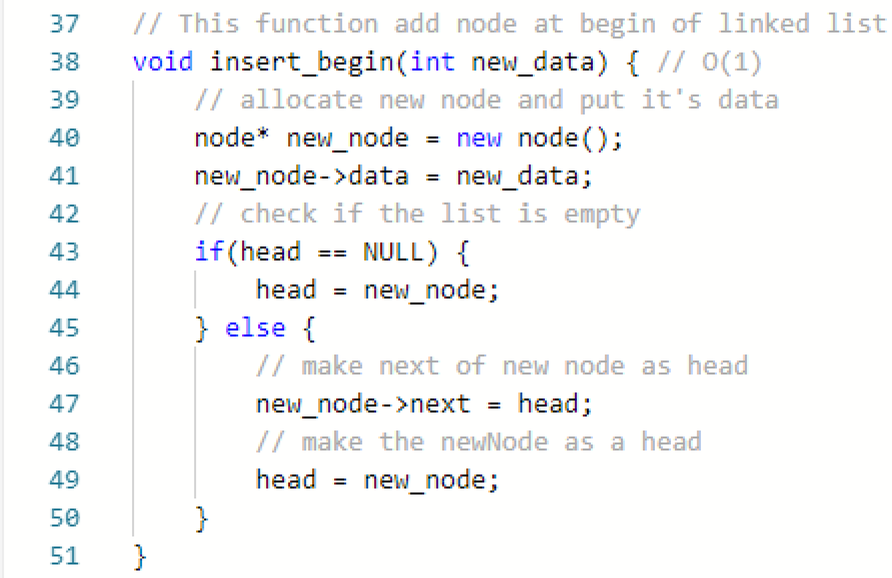


© Prepared by: Mohamed Ayman

Insertion Operation in C++

Insert node at begin of Linked List

Link: repl.it/repls/SmartSecondaryLanservers

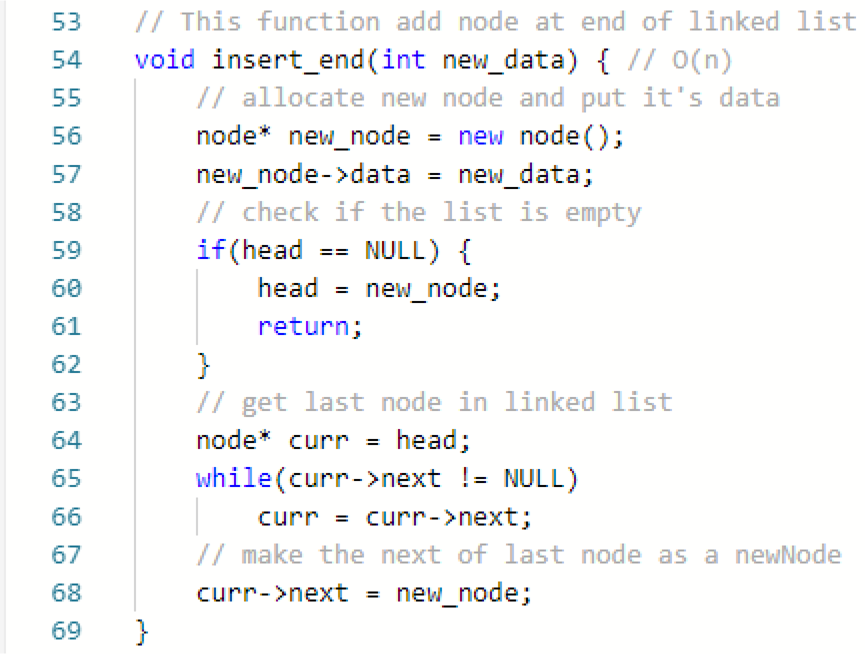


```cpp
// This function add node at begin of linked list
void insert_begin(int new_data) { // O(1)
    // allocate new node and put it's data
    node* new_node = new node();
    new_node->data = new_data;
    // check if the list is empty
    if(head == NULL) {
        head = new_node;
    } else {
        // make next of new node as head
        new_node->next = head;
        // make the newNode as a head
        head = new_node;
    }
}
```

Insertion Operation in C++

Insert node at end of Linked List

Link: repl.it/repls/SmartSecondaryLanservers

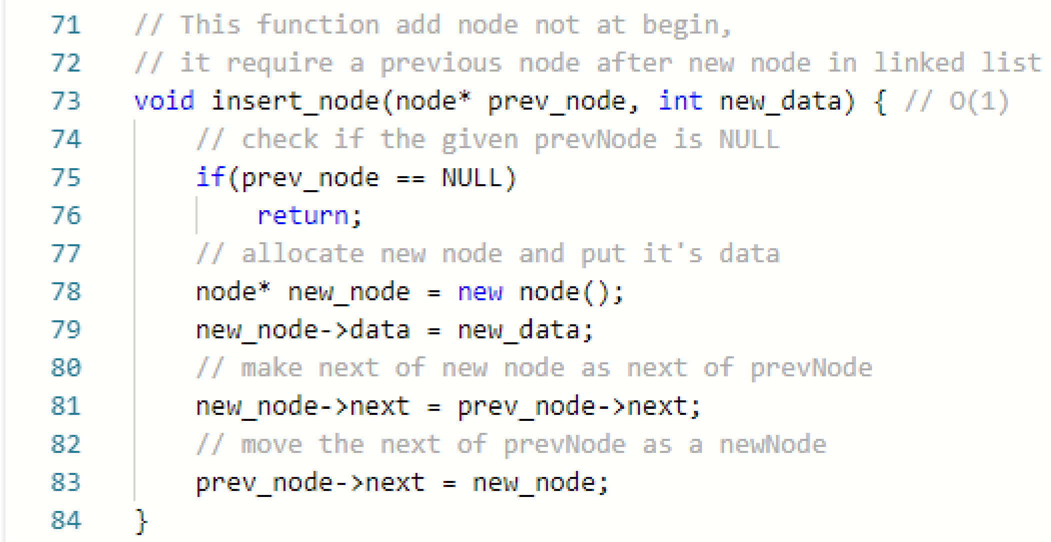


```cpp
// This function add node at end of linked list
void insert_end(int new_data) { // O(n)
    // allocate new node and put it's data
    node* new_node = new node();
    new_node->data = new_data;
    // check if the list is empty
    if(head == NULL) {
        head = new_node;
        return;
    }
    // get last node in linked list
    node* curr = head;
    while(curr->next != NULL)
        curr = curr->next;
    // make the next of last node as a newNode
    curr->next = new_node;
}
```

Insertion Operation in C++

Insert node in Linked List given previous node

Link: repl.it/repls/SmartSecondaryLanservers



```cpp
// This function add node not at begin,
// it require a previous node after new node in linked list
void insert_node(node* prev_node, int new_data) { // O(1)
    // check if the given prevNode is NULL
    if(prev_node == NULL)
        return;
    // allocate new node and put it's data
    node* new_node = new node();
    new_node->data = new_data;
    // make next of new node as next of prevNode
    new_node->next = prev_node->next;
    // move the next of prevNode as a newNode
    prev_node->next = new_node;
}
```

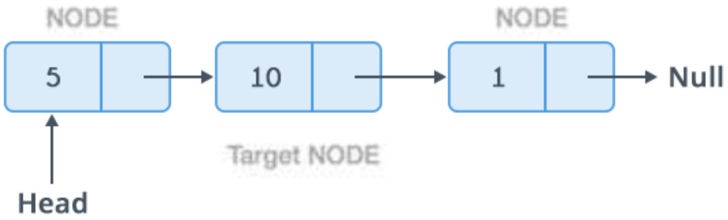
Agenda

- ✓ 1- Linked List Definition
- ✓ 2- Linked List Representation
- ✓ 3- Types of Linked List
- ✓ 4- Basic Operations
- ✓ 5- Insertion Operation
- 6- **Deletion Operation**
- 7- Traversal Operation
- 8- Search Operation

Deletion Operation

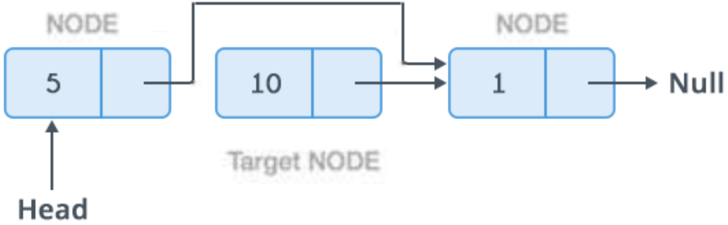
Deletion is also a more than one step process. We shall learn with pictorial representation.

First, locate the target node to be removed, by using searching algorithms.



The left (previous) node of the target node now should point to the next node

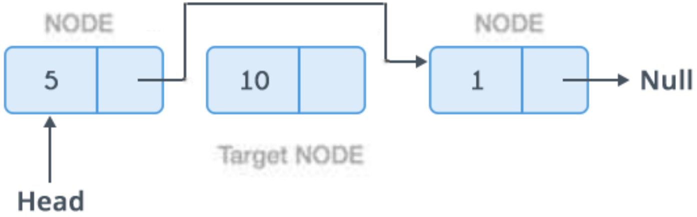
of the target node LeftNode.next –> TargetNode.next



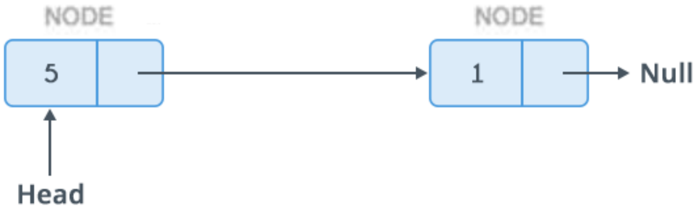
© Prepared by: Mohamed Ayman

Deletion Operation

This will remove the link that was pointing to the target node. Now, using the following code, we will remove what the target node is pointing at : TargetNode.next –› NULL



We need to use the deleted node. We can keep that in memory otherwise we can simply de-allocate memory and wipe off the target node completely.

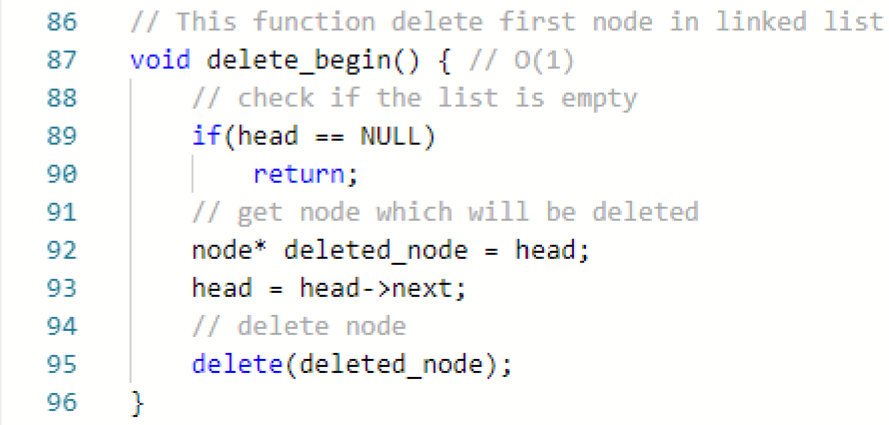


© Prepared by: Mohamed Ayman

Deletion Operation in C++

Delete node at begin of Linked List

Link: repl.it/repls/SmartSecondaryLanservers

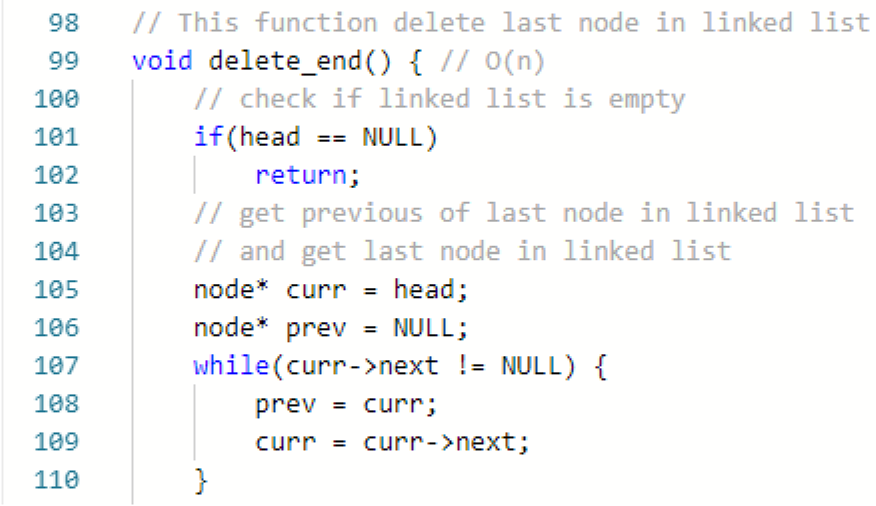


```cpp
// This function delete first node in linked list
void delete_begin() { // O(1)
    // check if the list is empty
    if(head == NULL)
        return;
    // get node which will be deleted
    node* deleted_node = head;
    head = head->next;
    // delete node
    delete(deleted_node);
}
```

Deletion Operation in C++

Delete node at end of Linked List

Link: repl.it/repls/SmartSecondaryLanservers



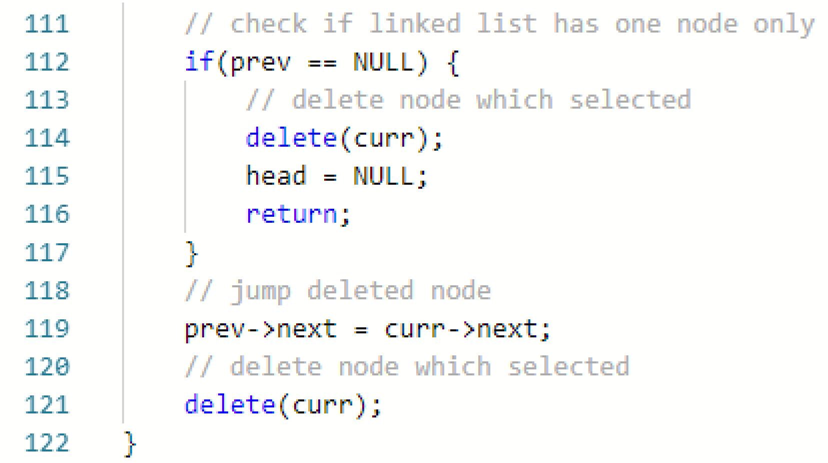
```
// This function delete last node in linked list
void delete_end() { // O(n)
    // check if linked list is empty
    if(head == NULL)
        return;
    // get previous of last node in linked list
    // and get last node in linked list
    node* curr = head;
    node* prev = NULL;
    while(curr->next != NULL) {
        prev = curr;
        curr = curr->next;
    }
```

© Prepared by: Mohamed Ayman

Deletion Operation in C++

Delete node at end of Linked List

Link: repl.it/repls/SmartSecondaryLanservers

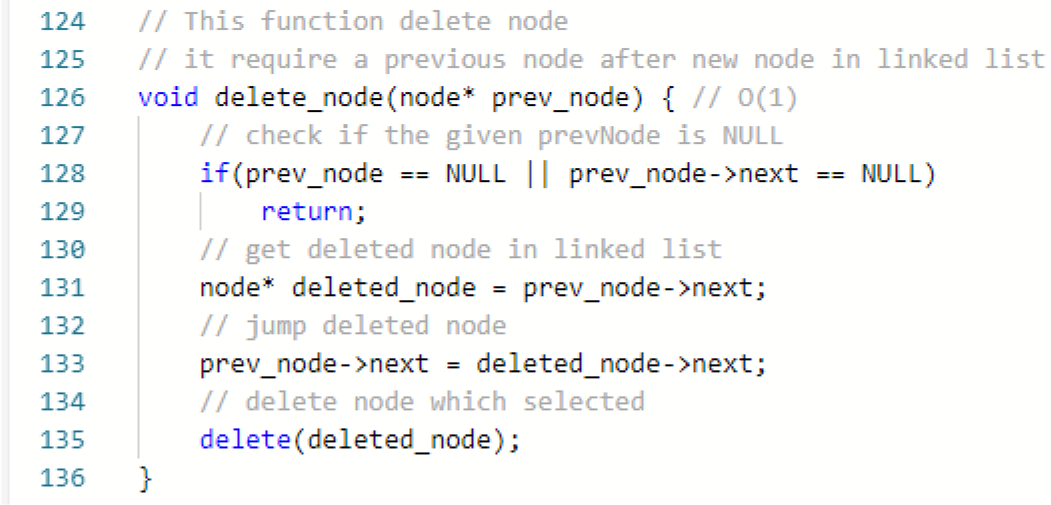


```
    // check if linked list has one node only
    if(prev == NULL) {
        // delete node which selected
        delete(curr);
        head = NULL;
        return;
    }
    // jump deleted node
    prev->next = curr->next;
    // delete node which selected
    delete(curr);
}
```

Deletion Operation in C++

Delete node in Linked List given previous node

Link: repl.it/repls/SmartSecondaryLanservers



```
// This function delete node
// it require a previous node after new node in linked list
void delete_node(node* prev_node) { // O(1)
    // check if the given prevNode is NULL
    if(prev_node == NULL || prev_node->next == NULL)
        return;
    // get deleted node in linked list
    node* deleted_node = prev_node->next;
    // jump deleted node
    prev_node->next = deleted_node->next;
    // delete node which selected
    delete(deleted_node);
}
```

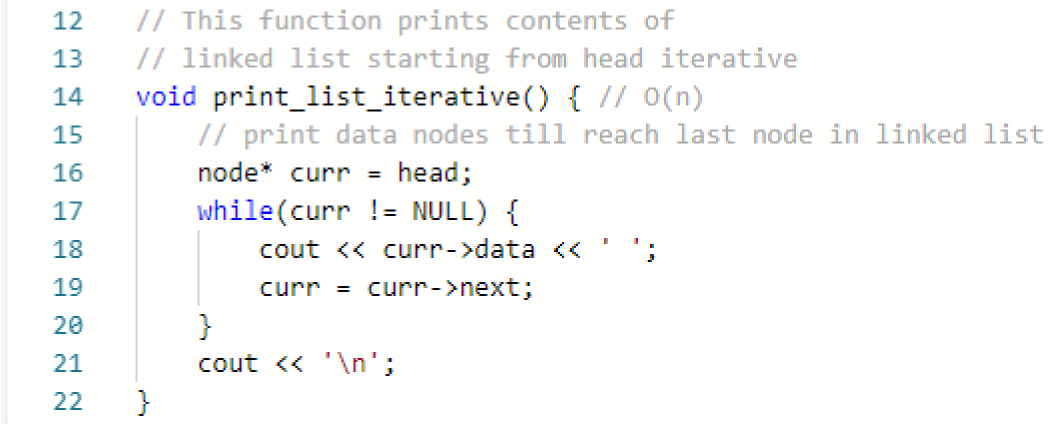
Agenda

- ✓ 1- Linked List Definition
- ✓ 2- Linked List Representation
- ✓ 3- Types of Linked List
- ✓ 4- Basic Operations
- ✓ 5- Insertion Operation
- ✓ 6- Deletion Operation
- 7- Traversal Operation
- 8- Search Operation

Traversal Operation

Print all nodes in Linked List in Iterative way

Link: repl.it/repls/SmartSecondaryLanservers



```
// This function prints contents of
// linked list starting from head iterative
void print_list_iterative() { // O(n)
    // print data nodes till reach last node in linked list
    node* curr = head;
    while(curr != NULL) {
        cout << curr->data << ' ';
        curr = curr->next;
    }
    cout << '\n';
}
```

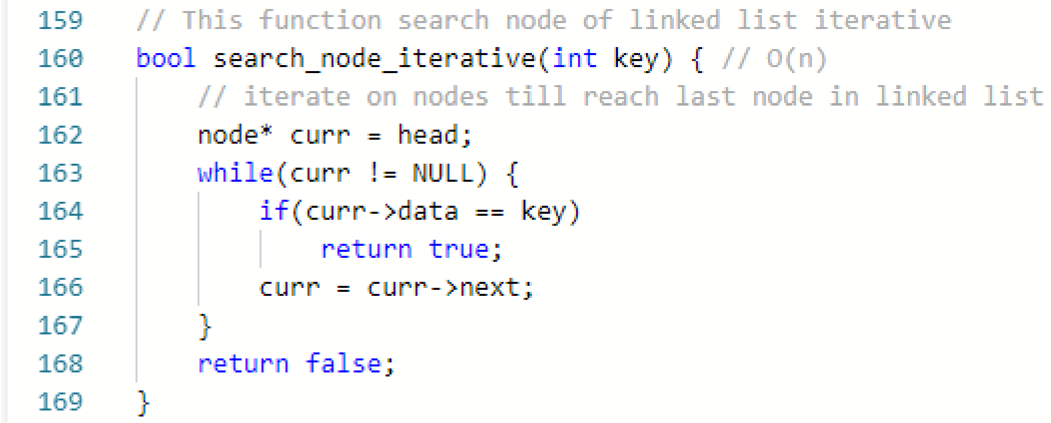
Agenda

- ✓ 1- Linked List Definition
- ✓ 2- Linked List Representation
- ✓ 3- Types of Linked List
- ✓ 4- Basic Operations
- ✓ 5- Insertion Operation
- ✓ 6- Deletion Operation
- ✓ 7- Traversal Operation
- 8- Search Operation

Search Operation

Search in Linked List in Iterative way

Link: repl.it/repls/SmartSecondaryLanservers



```
// This function search node of linked list iterative
bool search_node_iterative(int key) { // O(n)
    // iterate on nodes till reach last node in linked list
    node* curr = head;
    while(curr != NULL) {
        if(curr->data == key)
            return true;
        curr = curr->next;
    }
    return false;
}
```

Agenda

- ✓ 1- Linked List Definition
- ✓ 2- Linked List Representation
- ✓ 3- Types of Linked List
- ✓ 4- Basic Operations
- ✓ 5- Insertion Operation
- ✓ 6- Deletion Operation
- ✓ 7- Traversal Operation
- ✓ 8- Search Operation

DO MORE.

Practice

Practice

1- Count number of nodes in linked list by iterative way

2- Count number of nodes in linked list by recursive way

3- Print linked list by recursive way

4- Print linked list by iterative way

5- Print linked list in reversed order

6- Print element at middle of linked list

7- Print element at position i in linked list

8- Insert element at position i in linked list

9- Delete element at position i in linked list

10- Get node at position i in linked list

Practice

11- Delete all elements in linked list by recursive way

12- Delete all elements in linked list by iterative way

13- Search node in linked list by recursive way

14- Search node in linked list by iterative way

15- Swap any two nodes in linked list by index

16- Reverse linked list

17- Check if linked list has loop or not

18- Find length of loop in linked list

19- Remove duplication of nodes in sorted linked list

20- Remove duplication of nodes in unsorted linked list

Practice

21- Intersection of two sorted linked list

22- Intersection of two unsorted linked list

23- Union of two sorted linked list

24- Union of two unsorted linked list

25- Difference of two sorted linked list

26- Difference of two unsorted linked list

27- Segregate even and odd nodes in a linked list

28- Check if linked list is palindrome or not

29- Count number of times which element occurs in linked list by recursive way

30- Count number of times which element occurs in linked list by iterative way

Let's
STARTUP

Practice

1- Count number of nodes in linked list by iterative way

2- Count number of nodes in linked list by recursive way

3- Print linked list by recursive way

4- Print linked list by iterative way

5- Print linked list in reversed order

6- Print element at middle of linked list

7- Print element at position i in linked list

8- Insert element at position i in linked list

9- Delete element at position i in linked list

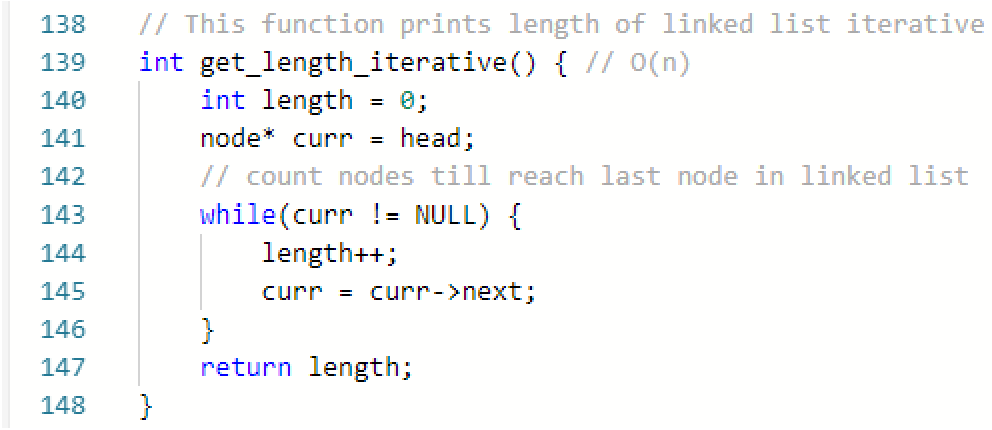
10- Get node at position i in linked list

Count number of nodes in linked list by iterative way

- Implement function which count number of nodes in linked list in iterative way
- Function Name: get length iterative
- Parameters: pointer of node to head of linked list => (node* curr)
- Return: int number of nodes in linked list

Count number of nodes in linked list by iterative way

Link: repl.it/repls/SmartSecondaryLanservers



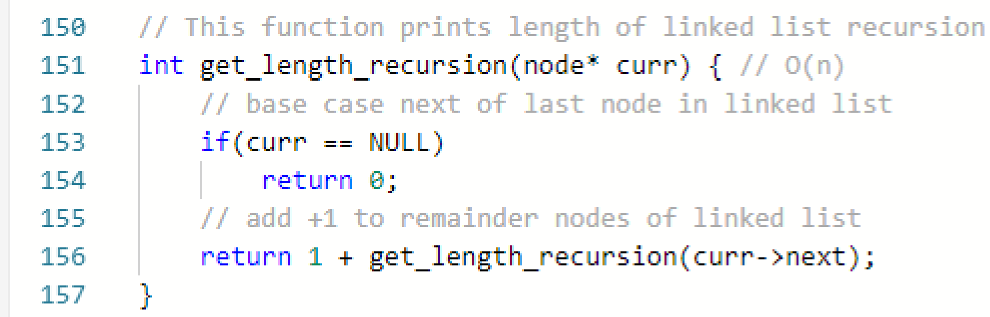
```cpp
// This function prints length of linked list iterative
int get_length_iterative() { // O(n)
    int length = 0;
    node* curr = head;
    // count nodes till reach last node in linked list
    while(curr != NULL) {
        length++;
        curr = curr->next;
    }
    return length;
}
```

Count number of nodes in linked list by recursive way

- Implement function which count number of nodes in linked list in recursive way
- Function Name: get length recursion
- Parameters: pointer of node to head of linked list => (node* curr)
- Return: int number of nodes in linked list

Count number of nodes in linked list by recursive way

Link: repl.it/repls/SmartSecondaryLanservers



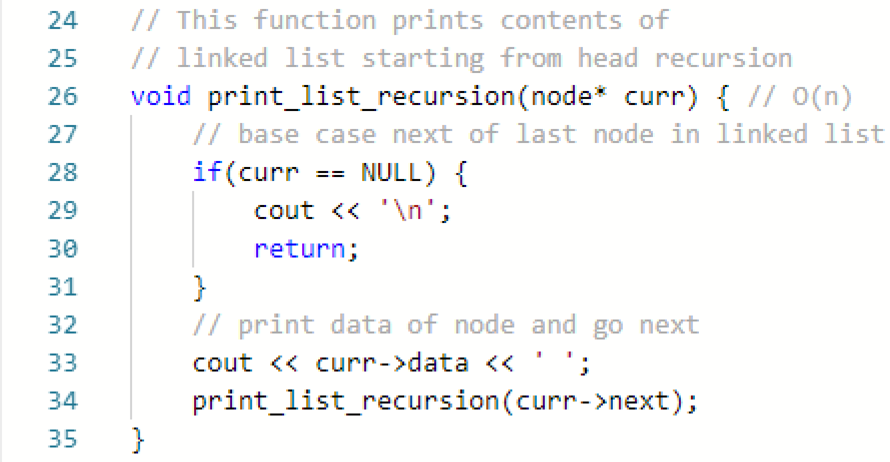
```cpp
// This function prints length of linked list recursion
int get_length_recursion(node* curr) { // O(n)
    // base case next of last node in linked list
    if(curr == NULL)
        return 0;
    // add +1 to remainder nodes of linked list
    return 1 + get_length_recursion(curr->next);
}
```

Print linked list by recursive way

- Implement function which print nodes of linked list in recursive way
- Function Name: print list recursion
- Parameters: pointer of node to head of linked list => (node* curr)
- Return: None

Print linked list by recursive way

Link: repl.it/repls/SmartSecondaryLanservers



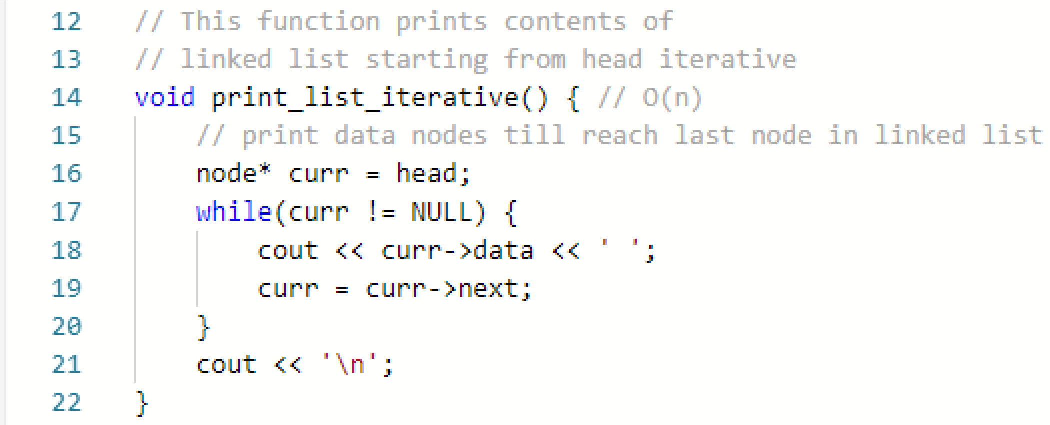
```cpp
// This function prints contents of
// linked list starting from head recursion
void print_list_recursion(node* curr) { // O(n)
    // base case next of last node in linked list
    if(curr == NULL) {
        cout << '\n';
        return;
    }
    // print data of node and go next
    cout << curr->data << ' ';
    print_list_recursion(curr->next);
}
```

Print linked list by iterative way

- Implement function which print nodes of linked list in iterative way
- Function Name: print list iterative
- Parameters: pointer of node to head of linked list => (node* curr)
- Return: None

Print linked list by iterative way

Link: repl.it/repls/SmartSecondaryLanservers



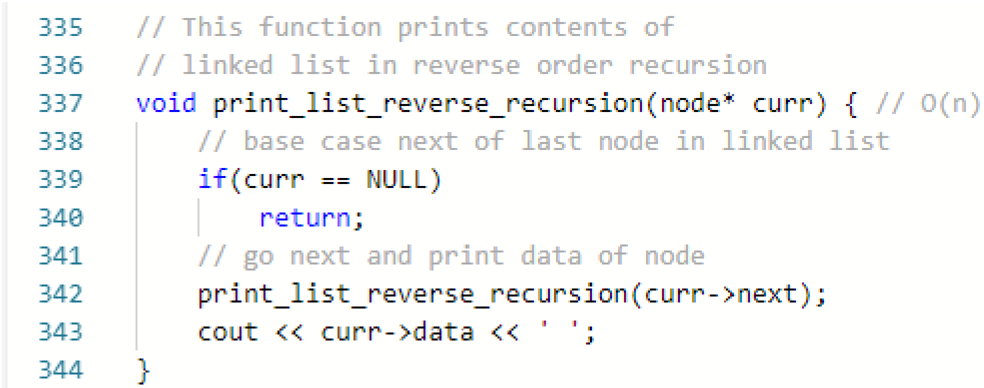
```cpp
// This function prints contents of
// linked list starting from head iterative
void print_list_iterative() { // O(n)
    // print data nodes till reach last node in linked list
    node* curr = head;
    while(curr != NULL) {
        cout << curr->data << ' ';
        curr = curr->next;
    }
    cout << '\n';
}
```

Print linked list in reversed order

- Implement function which print nodes of linked list in reverse order by recursive way
- Function Name: print list reverse recursive
- Parameters: pointer of node to head of linked list => (node* curr)
- Return: None

Print linked list in reversed order

Link: repl.it/repls/SmartSecondaryLanservers



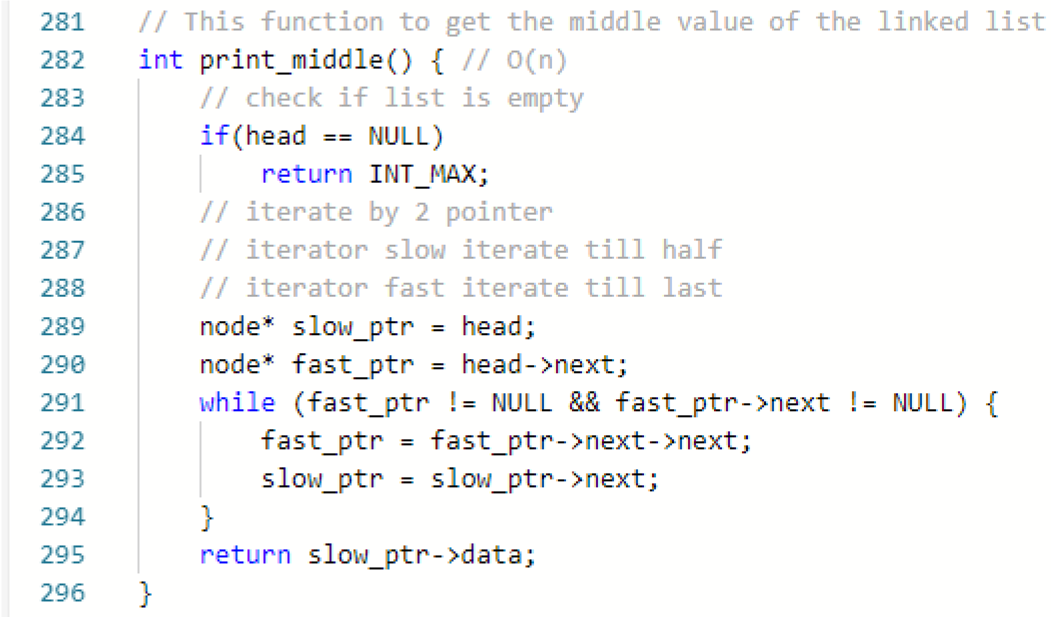
```cpp
// This function prints contents of
// linked list in reverse order recursion
void print_list_reverse_recursion(node* curr) { // O(n)
    // base case next of last node in linked list
    if(curr == NULL)
        return;
    // go next and print data of node
    print_list_reverse_recursion(curr->next);
    cout << curr->data << ' ';
}
```

Print element at middle of linked list

- Implement function which print the middle node of linked list
- Function Name: print middle
- Parameters: None
- Return: pointer of node

Print element at middle of linked list

Link: repl.it/repls/SmartSecondaryLanservers



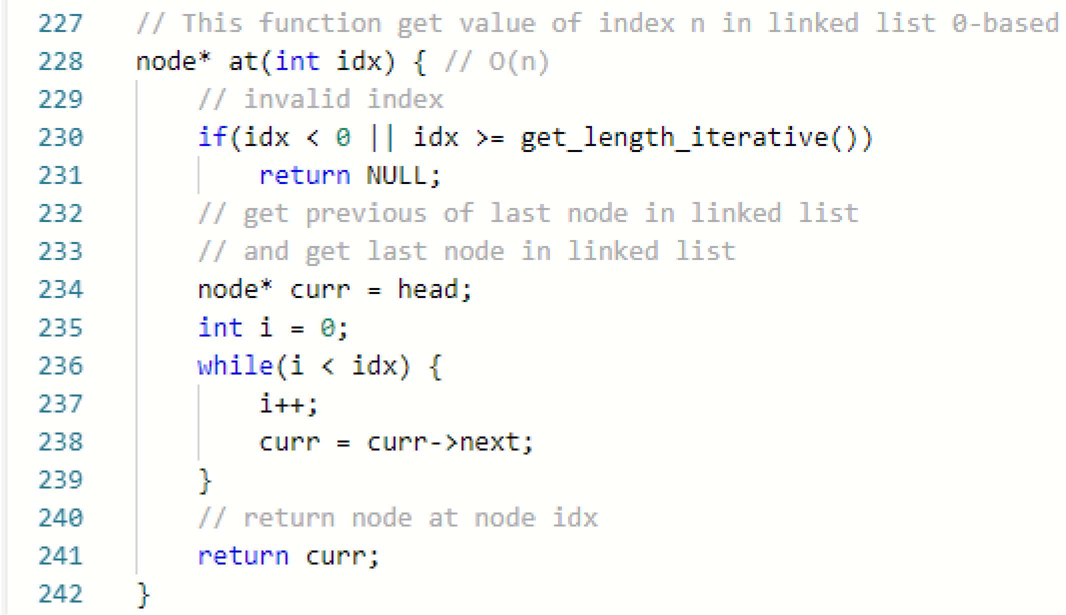
```
// This function to get the middle value of the linked list
int print_middle() { // O(n)
    // check if list is empty
    if(head == NULL)
        return INT_MAX;
    // iterate by 2 pointer
    // iterator slow iterate till half
    // iterator fast iterate till last
    node* slow_ptr = head;
    node* fast_ptr = head->next;
    while (fast_ptr != NULL && fast_ptr->next != NULL) {
        fast_ptr = fast_ptr->next->next;
        slow_ptr = slow_ptr->next;
    }
    return slow_ptr->data;
}
```

Print element at position i in linked list

- Implement function which print node at position i of linked list
- Function Name: at
- Parameters: int idx => present specific index
- Return: pointer of node

Print element at position i in linked list

Link: repl.it/repls/SmartSecondaryLanservers



```
// This function get value of index n in linked list 0-based
node* at(int idx) { // O(n)
    // invalid index
    if(idx < 0 || idx >= get_length_iterative())
        return NULL;
    // get previous of last node in linked list
    // and get last node in linked list
    node* curr = head;
    int i = 0;
    while(i < idx) {
        i++;
        curr = curr->next;
    }
    // return node at node idx
    return curr;
}
```

Insert element at position i in linked list

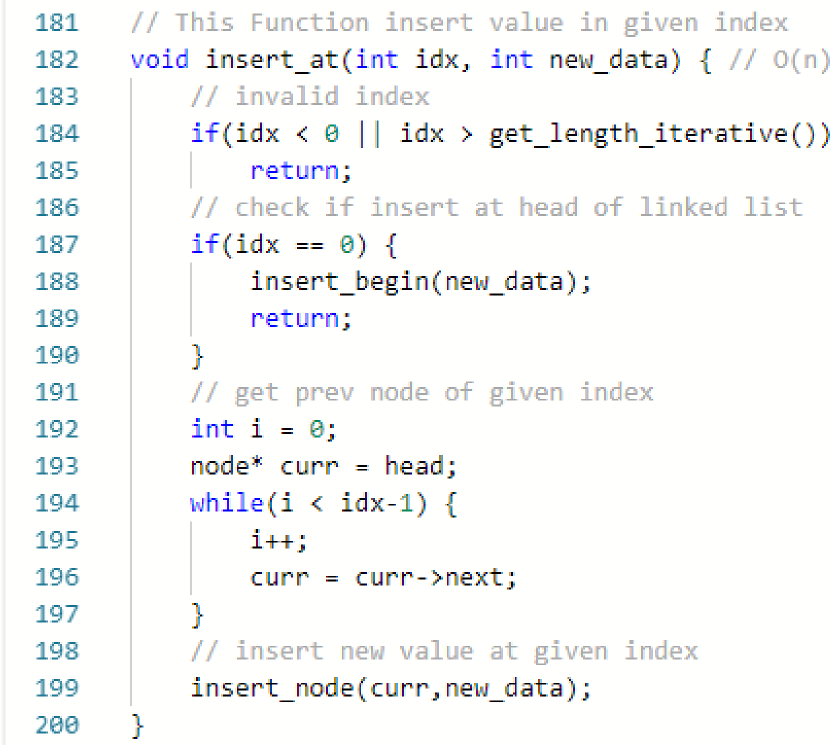
- Implement function which add new node in specific valid position in linked list
- Function Name: insert at
- Parameters: (idx, new data)

  int idx => it mean a position to insert at it

  int new data => it mean data of new node
- Return: None

Insert element at position i in linked list

Link: repl.it/repls/SmartSecondaryLanservers



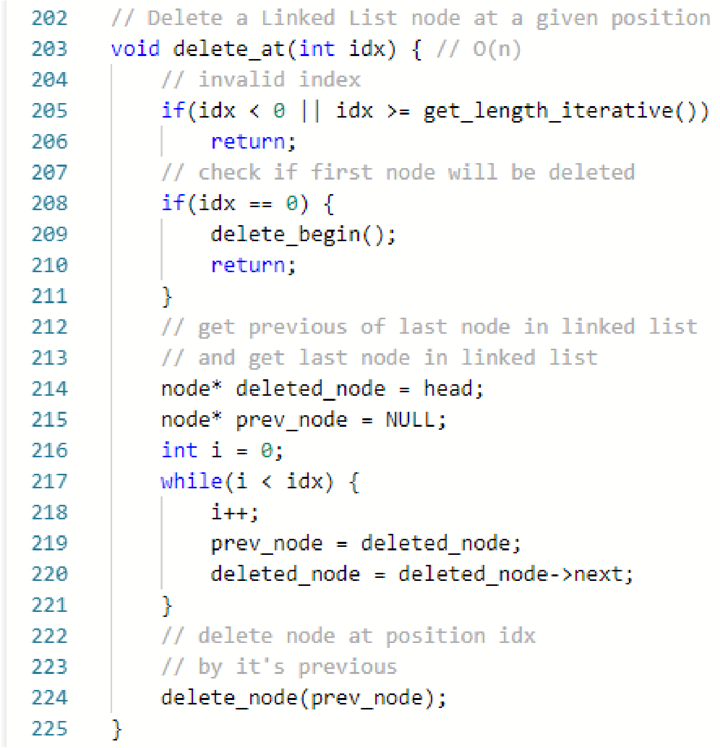
```
// This Function insert value in given index
void insert_at(int idx, int new_data) { // O(n)
    // invalid index
    if(idx < 0 || idx > get_length_iterative())
        return;
    // check if insert at head of linked list
    if(idx == 0) {
        insert_begin(new_data);
        return;
    }
    // get prev node of given index
    int i = 0;
    node* curr = head;
    while(i < idx-1) {
        i++;
        curr = curr->next;
    }
    // insert new value at given index
    insert_node(curr,new_data);
}
```

Delete element at position i in linked list

- Implement function which delete node in specific valid position in linked list
- Function Name: delete at
- Parameters: int idx => it mean a position to delete at it
- Return: None

Delete element at position i in linked list

Link: repl.it/repls/SmartSecondaryLanservers



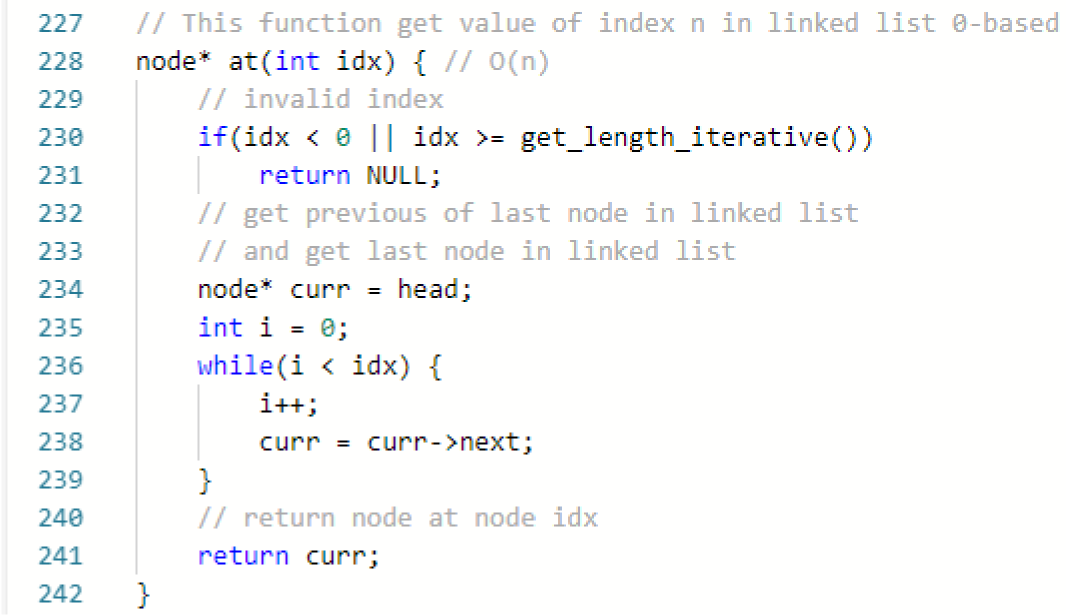
```cpp
// Delete a Linked List node at a given position
void delete_at(int idx) { // O(n)
    // invalid index
    if(idx < 0 || idx >= get_length_iterative())
        return;
    // check if first node will be deleted
    if(idx == 0) {
        delete_begin();
        return;
    }
    // get previous of last node in linked list
    // and get last node in linked list
    node* deleted_node = head;
    node* prev_node = NULL;
    int i = 0;
    while(i < idx) {
        i++;
        prev_node = deleted_node;
        deleted_node = deleted_node->next;
    }
    // delete node at position idx
    // by it's previous
    delete_node(prev_node);
}
```

Get node at position i in linked list

- Implement function which get node at position i of linked list
- Function Name: at
- Parameters: int idx => present specific index
- Return: pointer of node

Get node at position i in linked list

Link: repl.it/repls/SmartSecondaryLanservers



```cpp
// This function get value of index n in linked list 0-based
node* at(int idx) { // O(n)
    // invalid index
    if(idx < 0 || idx >= get_length_iterative())
        return NULL;
    // get previous of last node in linked list
    // and get last node in linked list
    node* curr = head;
    int i = 0;
    while(i < idx) {
        i++;
        curr = curr->next;
    }
    // return node at node idx
    return curr;
}
```

Practice

~~1- Count number of nodes in linked list by iterative way~~

~~2- Count number of nodes in linked list by recursive way~~

~~3- Print linked list by recursive way~~

~~4- Print linked list by iterative way~~

~~5- Print linked list in reversed order~~

~~6- Print element at middle of linked list~~

~~7- Print element at position i in linked list~~

~~8- Insert element at position i in linked list~~

~~9- Delete element at position i in linked list~~

~~10- Get node at position i in linked list~~

DO
MORE.

Practice

11- Delete all elements in linked list by recursive way

12- Delete all elements in linked list by iterative way

13- Search node in linked list by recursive way

14- Search node in linked list by iterative way

15- Swap any two nodes in linked list by index

16- Reverse linked list

17- Check if linked list has loop or not

18- Find length of loop in linked list

19- Remove duplication of nodes in sorted linked list

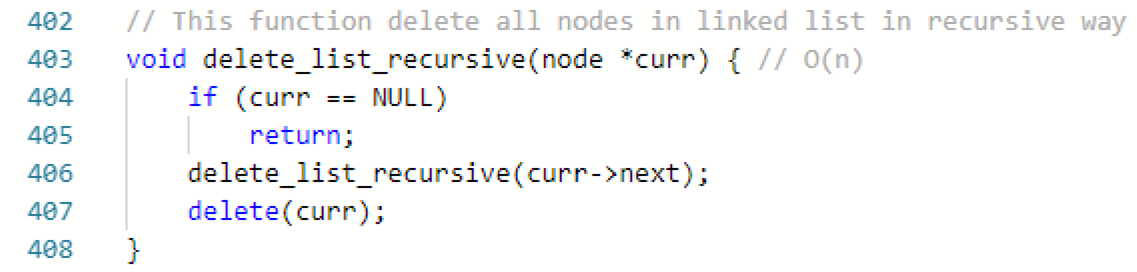
20- Remove duplication of nodes in unsorted linked list

Delete all elements in linked list by recursive way

- Implement function which delete all nodes of linked list in recursive way
- Function Name: delete list recursive
- Parameters: pointer of node to head of linked list => (node* curr)
- Return: None

Delete all elements in linked list by recursive way

Link: repl.it/repls/SmartSecondaryLanservers



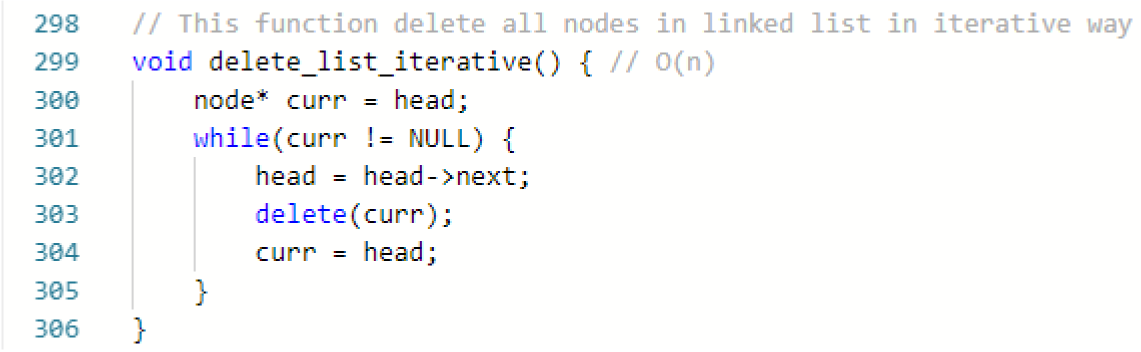
```
// This function delete all nodes in linked list in recursive way
void delete_list_recursive(node *curr) { // O(n)
    if (curr == NULL)
        return;
    delete_list_recursive(curr->next);
    delete(curr);
}
```

Delete all elements in linked list by iterative way

- Implement function which delete all nodes of linked list in iterative way
- Function Name: delete list iterative
- Parameters: None
- Return: None

Delete all elements in linked list by iterative way

Link: repl.it/repls/SmartSecondaryLanservers



```
// This function delete all nodes in linked list in iterative way
void delete_list_iterative() { // O(n)
    node* curr = head;
    while(curr != NULL) {
        head = head->next;
        delete(curr);
        curr = head;
    }
}
```

Search node in linked list by recursive way

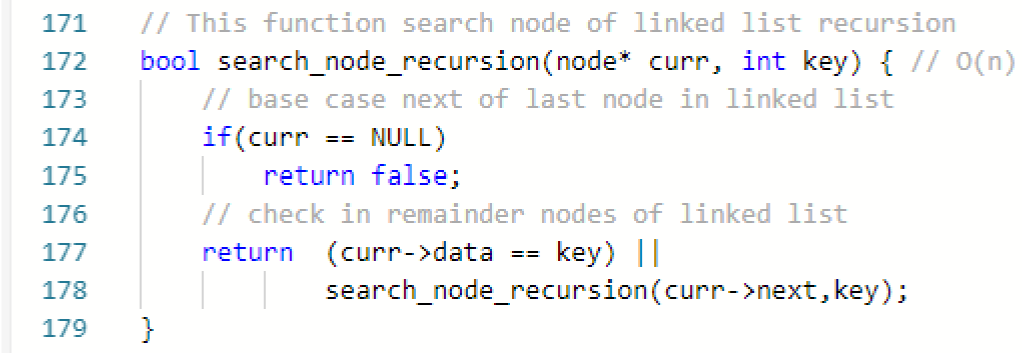
- Implement function which search about node of linked list in recursive way
- Function Name: search node recursion
- Parameters: (node* curr, int key)

  node* curr => it mean curr node in linked list, it will start with head

  key => it mean key of node which check this node in linked list or not
- Return: boolean value, True if this key has been found in linked list or False otherwise

Search node in linked list by recursive way

Link: repl.it/repls/SmartSecondaryLanservers



```cpp
// This function search node of linked list recursion
bool search_node_recursion(node* curr, int key) { // O(n)
    // base case next of last node in linked list
    if(curr == NULL)
        return false;
    // check in remainder nodes of linked list
    return  (curr->data == key) ||
            search_node_recursion(curr->next,key);
}
```

Search node in linked list by iterative way

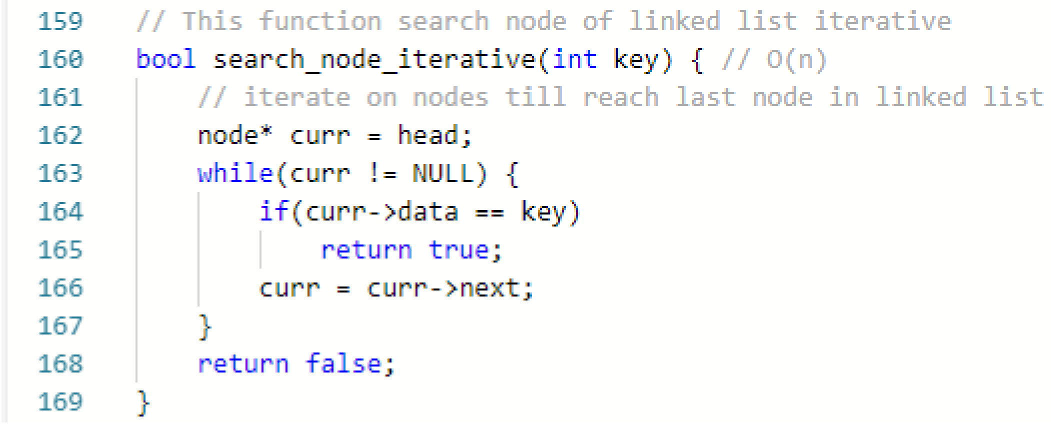
- Implement function which search about node of linked list in iterative way
- Function Name: search node iterative
- Parameters: (node* curr, int key)

  node* curr => it mean curr node in linked list, it will start with head

  key => it mean key of node which check this node in linked list or not
- Return: boolean value, True if this key has been found in linked list or False otherwise

Search node in linked list by iterative way

Link: repl.it/repls/SmartSecondaryLanservers



```
// This function search node of linked list iterative
bool search_node_iterative(int key) { // O(n)
    // iterate on nodes till reach last node in linked list
    node* curr = head;
    while(curr != NULL) {
        if(curr->data == key)
            return true;
        curr = curr->next;
    }
    return false;
}
```

Swap any two nodes in linked list by index

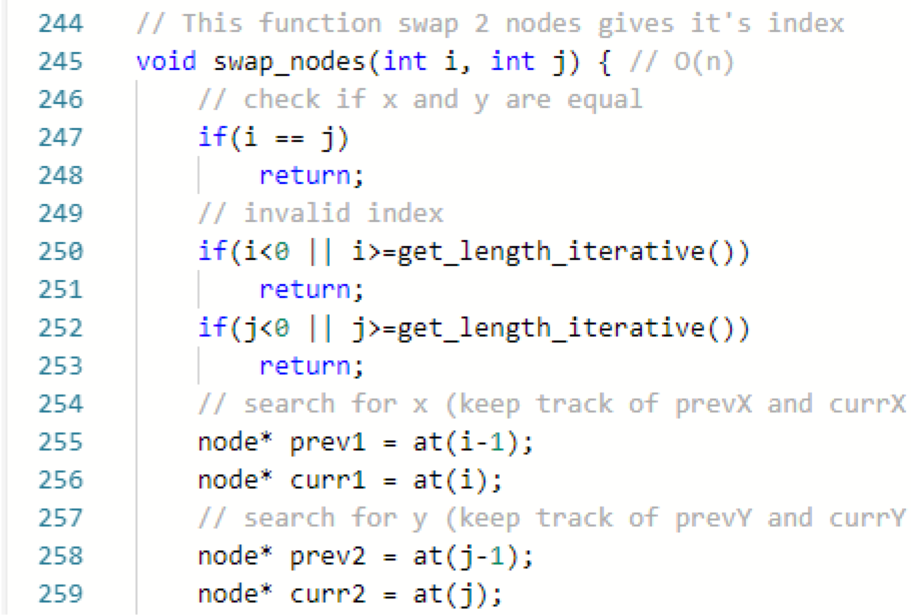
- Implement function swap two nodes given their indices
- Function Name: swap nodes
- Parameters: (i, j)

  i =›index of node

  j =› index of other node
- Return: None

Swap any two nodes in linked list by index

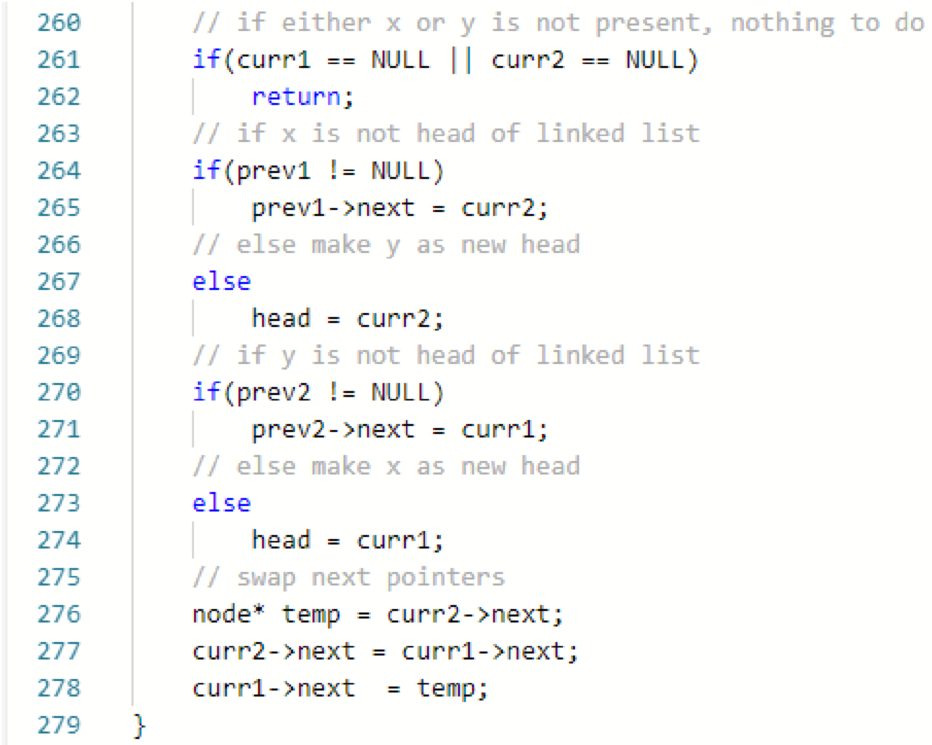
Link: repl.it/repls/SmartSecondaryLanservers



```
// This function swap 2 nodes gives it's index
void swap_nodes(int i, int j) { // O(n)
    // check if x and y are equal
    if(i == j)
        return;
    // invalid index
    if(i<0 || i>=get_length_iterative())
        return;
    if(j<0 || j>=get_length_iterative())
        return;
    // search for x (keep track of prevX and currX
    node* prev1 = at(i-1);
    node* curr1 = at(i);
    // search for y (keep track of prevY and currY
    node* prev2 = at(j-1);
    node* curr2 = at(j);
```

Swap any two nodes in linked list by index

Link: repl.it/repls/SmartSecondaryLanservers

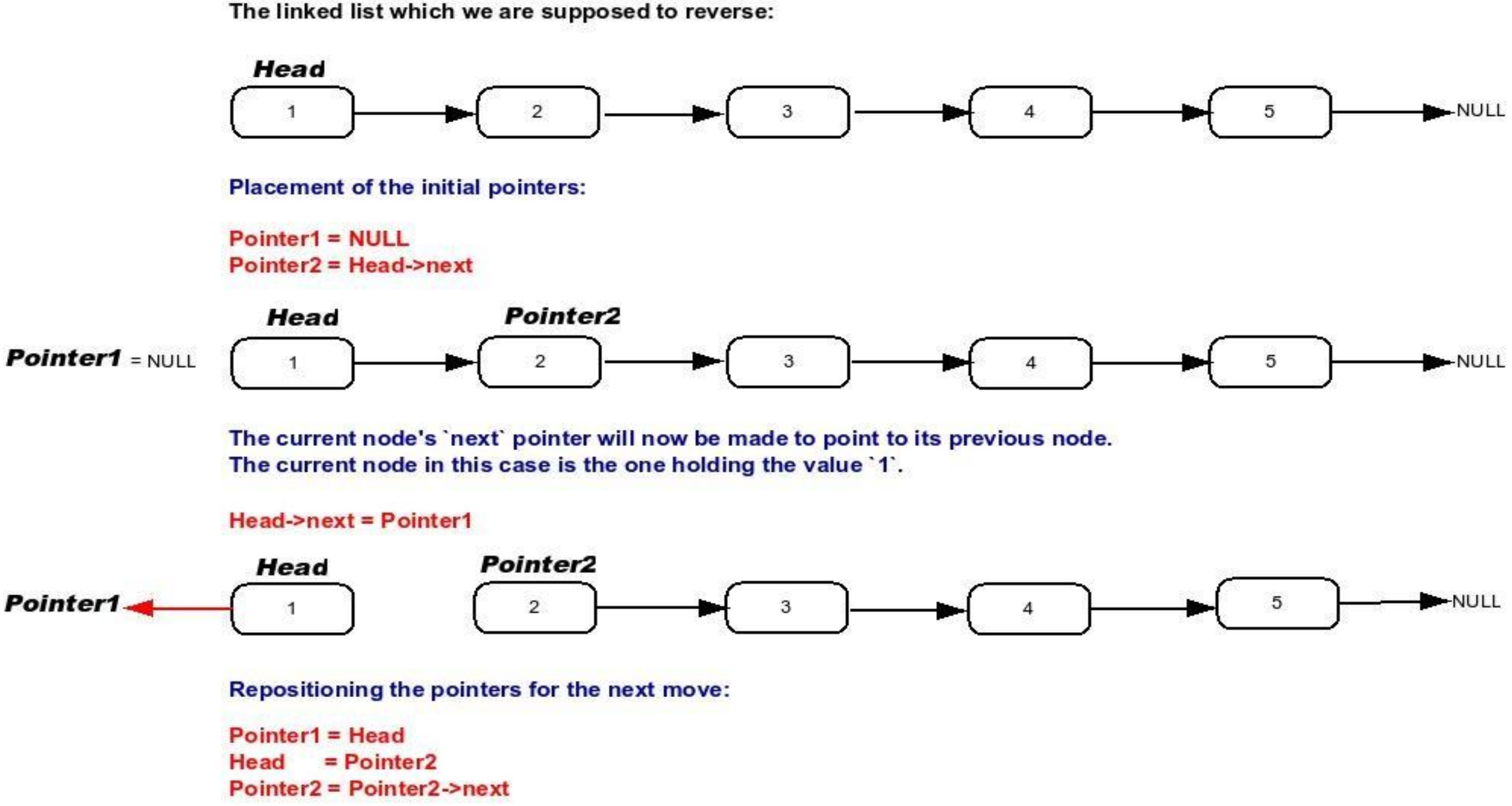


```
    // if either x or y is not present, nothing to do
    if(curr1 == NULL || curr2 == NULL)
        return;
    // if x is not head of linked list
    if(prev1 != NULL)
        prev1->next = curr2;
    // else make y as new head
    else
        head = curr2;
    // if y is not head of linked list
    if(prev2 != NULL)
        prev2->next = curr1;
    // else make x as new head
    else
        head = curr1;
    // swap next pointers
    node* temp = curr2->next;
    curr2->next = curr1->next;
    curr1->next  = temp;
}
```

Reverse linked list

- Implement function reverse linked list
- Function Name: reverse
- Parameters: None
- Return: None

Reverse linked list



The linked list which we are supposed to reverse:

Placement of the initial pointers:

```
Pointer1 = NULL
Pointer2 = Head->next
```

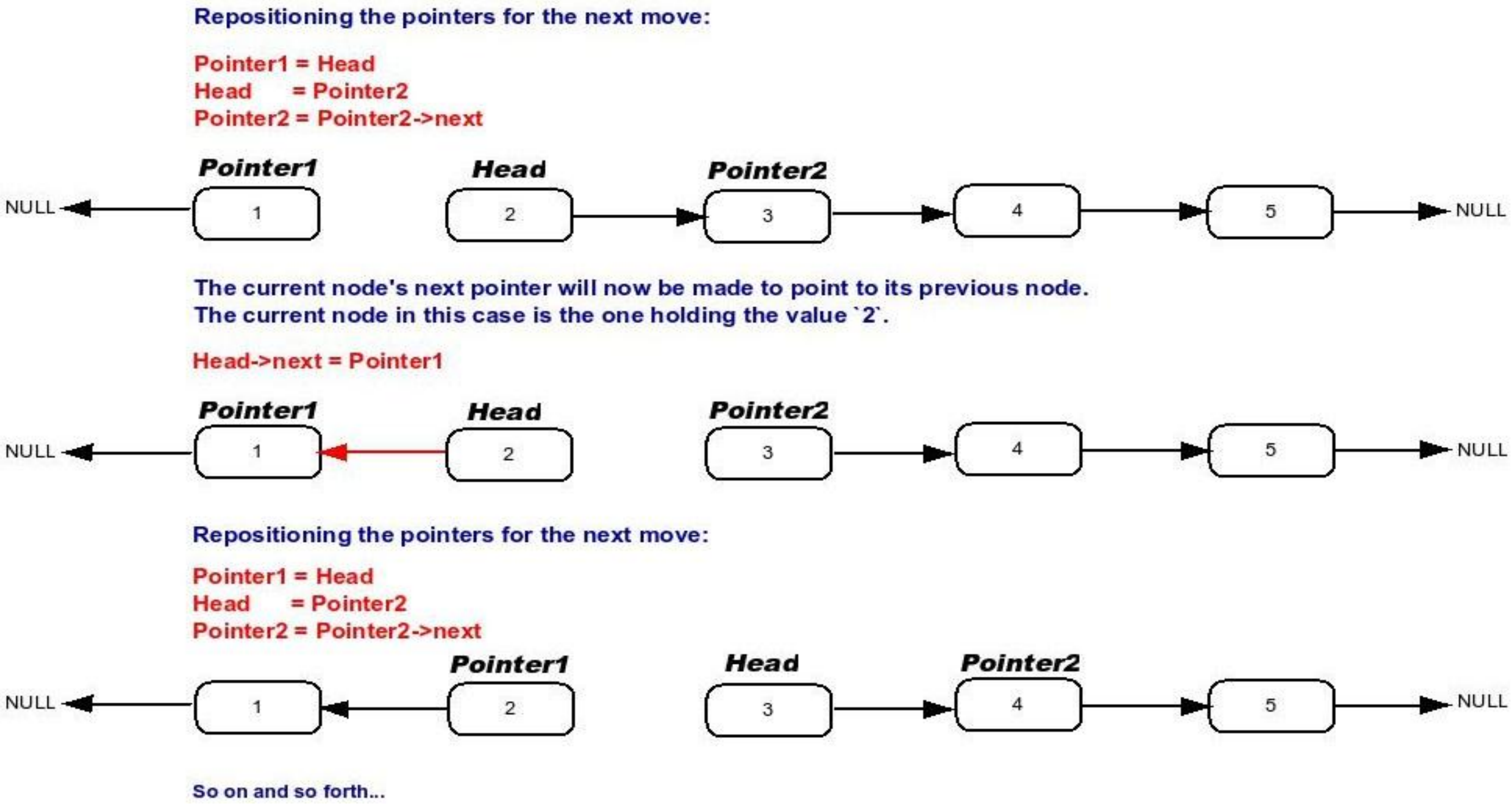
The current node's `next` pointer will now be made to point to its previous node.
The current node in this case is the one holding the value `1`.

```
Head->next = Pointer1
```

Repositioning the pointers for the next move:

```
Pointer1 = Head
Head     = Pointer2
Pointer2 = Pointer2->next
```

Reverse linked list



Repositioning the pointers for the next move:

```
Pointer1 = Head
Head     = Pointer2
Pointer2 = Pointer2->next
```

The current node's next pointer will now be made to point to its previous node.
The current node in this case is the one holding the value `2`.

```
Head->next = Pointer1
```

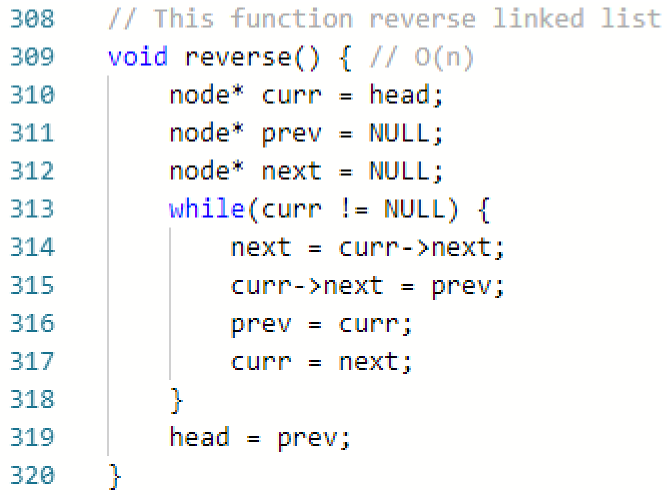
Repositioning the pointers for the next move:

```
Pointer1 = Head
Head     = Pointer2
Pointer2 = Pointer2->next
```

So on and so forth...

Reverse linked list

Link: repl.it/repls/SmartSecondaryLanservers

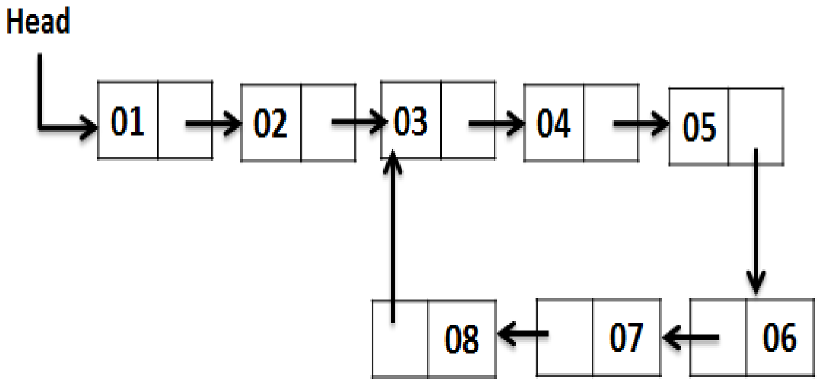


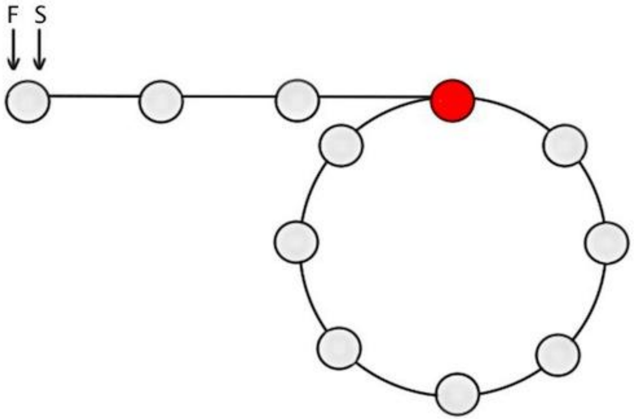
```
// This function reverse linked list
void reverse() { // O(n)
    node* curr = head;
    node* prev = NULL;
    node* next = NULL;
    while(curr != NULL) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    head = prev;
}
```

Check if linked list has loop or not

- Implement function which detect if linked list has a cycle (loop) or not
- Function Name: detect loop
- Parameters: None
- Return: boolean value, True if linked list has a cycle otherwise False

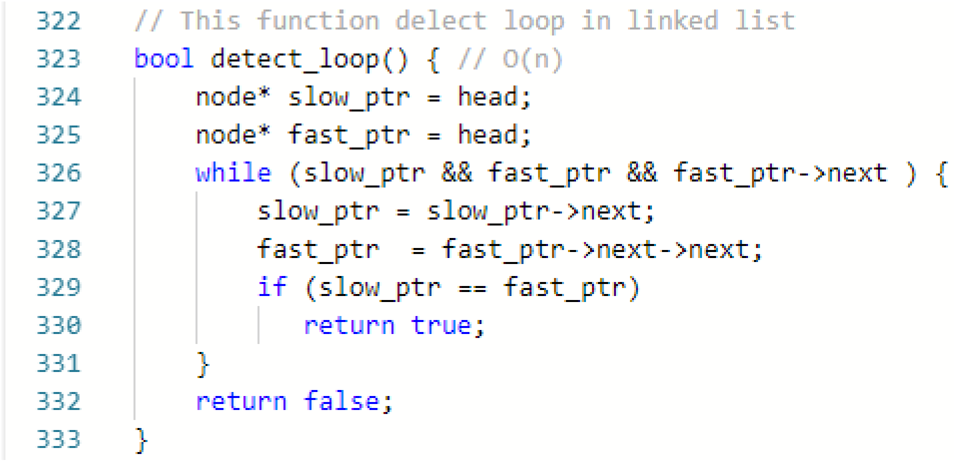
Check if linked list has loop or not





Check if linked list has loop or not

Link: repl.it/repls/SmartSecondaryLanservers



```
// This function delect loop in linked list
bool detect_loop() { // O(n)
    node* slow_ptr = head;
    node* fast_ptr = head;
    while (slow_ptr && fast_ptr && fast_ptr->next ) {
        slow_ptr = slow_ptr->next;
        fast_ptr  = fast_ptr->next->next;
        if (slow_ptr == fast_ptr)
            return true;
    }
    return false;
}
```

Find length of loop in linked list

- Implement function which calculate length of cycle in linked list
- Function Name: cycle length
- Parameters: None
- Return: None

Remove duplication of nodes in sorted linked list

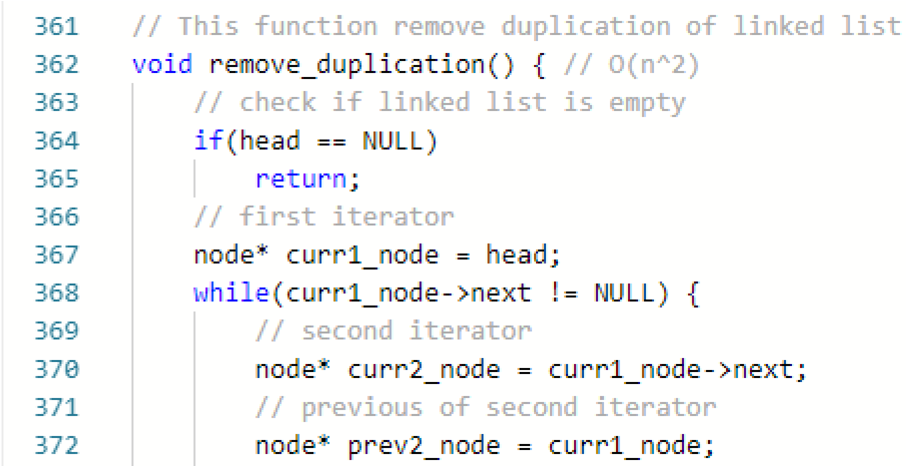
- Implement function which remove duplication of nodes in sorted linked list
- Function Name: remove duplication
- Parameters: None
- Return: None

Remove duplication of nodes in unsorted linked list

- Implement function which remove duplication of nodes in unsorted linked list
- Function Name: remove duplication
- Parameters: None
- Return: None

Remove duplication of nodes in unsorted linked list

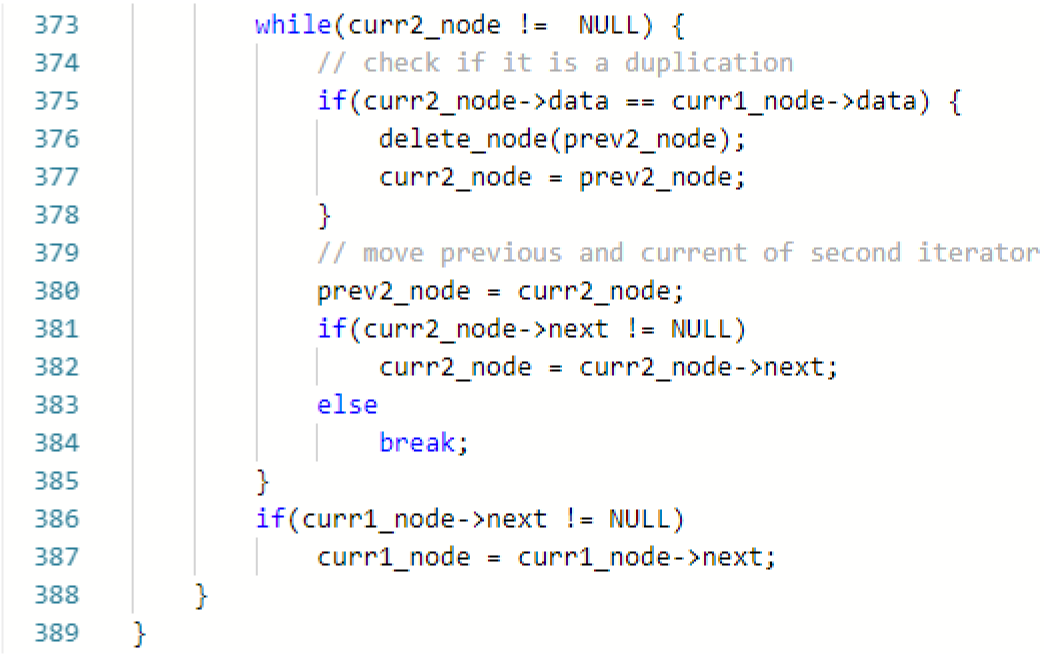
Link: repl.it/repls/SmartSecondaryLanservers



```
// This function remove duplication of linked list
void remove_duplication() { // O(n^2)
    // check if linked list is empty
    if(head == NULL)
        return;
    // first iterator
    node* curr1_node = head;
    while(curr1_node->next != NULL) {
        // second iterator
        node* curr2_node = curr1_node->next;
        // previous of second iterator
        node* prev2_node = curr1_node;
```

Remove duplication of nodes in unsorted linked list

Link: repl.it/repls/SmartSecondaryLanservers



```
        while(curr2_node !=  NULL) {
            // check if it is a duplication
            if(curr2_node->data == curr1_node->data) {
                delete_node(prev2_node);
                curr2_node = prev2_node;
            }
            // move previous and current of second iterator
            prev2_node = curr2_node;
            if(curr2_node->next != NULL)
                curr2_node = curr2_node->next;
            else
                break;
        }
        if(curr1_node->next != NULL)
            curr1_node = curr1_node->next;
    }
}
```

Practice

~~11- Delete all elements in linked list by recursive way~~

~~12- Delete all elements in linked list by iterative way~~

~~13- Search node in linked list by recursive way~~

~~14- Search node in linked list by iterative way~~

~~15- Swap any two nodes in linked list by index~~

~~16- Reverse linked list~~

~~17- Check if linked list has loop or not~~

~~18- Find length of loop in linked list~~

~~19- Remove duplication of nodes in sorted linked list~~

~~20- Remove duplication of nodes in unsorted linked list~~

DO
MORE.

Practice

21- Intersection of two sorted linked list

22- Intersection of two unsorted linked list

23- Union of two sorted linked list

24- Union of two unsorted linked list

25- Difference of two sorted linked list

26- Difference of two unsorted linked list

27- Segregate even and odd nodes in a linked list

28- Check if linked list is palindrome or not

29- Count number of times which element occurs in linked list by recursive way

30- Count number of times which element occurs in linked list by iterative way

Intersection of two sorted linked list

- Implement function which get intersection nodes of two sorted linked list
- Function Name: intersection
- Parameters: pointer of node to head of first linked list => (node* curr1)

  pointer of node to head of second linked list => (node* curr2)
- Return: pointer of node to head of result linked list

Intersection of two unsorted linked list

- Implement function which get intersection nodes of two unsorted linked list
- Function Name: intersection
- Parameters: pointer of node to head of first linked list => (node* curr1)

  pointer of node to head of second linked list => (node* curr2)
- Return: pointer of node to head of result linked list

© Prepared by: Mohamed Ayman

Union of two sorted linked list

- Implement function which get union nodes of two sorted linked list
- Function Name: union
- Parameters: pointer of node to head of first linked list => (node* curr1)

  pointer of node to head of second linked list => (node* curr2)
- Return: pointer of node to head of result linked list

Union of two unsorted linked list

- Implement function which get union nodes of two unsorted linked list
- Function Name: union
- Parameters: pointer of node to head of first linked list => (node* curr1)

  pointer of node to head of second linked list => (node* curr2)
- Return: pointer of node to head of result linked list

Difference of two sorted linked list

- Implement function which get difference nodes of two sorted linked list
- Function Name: difference
- Parameters: pointer of node to head of first linked list => (node* curr1)

  pointer of node to head of second linked list => (node* curr2)
- Return: pointer of node to head of result linked list

Difference of two unsorted linked list

- Implement function which get difference nodes of two unsorted linked list
- Function Name: difference
- Parameters: pointer of node to head of first linked list => (node* curr1)

  pointer of node to head of second linked list => (node* curr2)
- Return: pointer of node to head of result linked list

Segregate even and odd nodes in a linked list

- Implement function to modify the linked list such that all even numbers appear before all the odd numbers in the modified linked list. Also, keep the order of even and odd numbers same.
- Function Name: segregate
- Parameters: pointer of node to head of linked list => (node* curr)
- Return: None

Check if linked list is palindrome or not

- Implement function which check if linked list is palindrome or not
- Function Name: is palindrome
- Parameters: None
- Return: boolean value, True if linked list is palindrome otherwise False

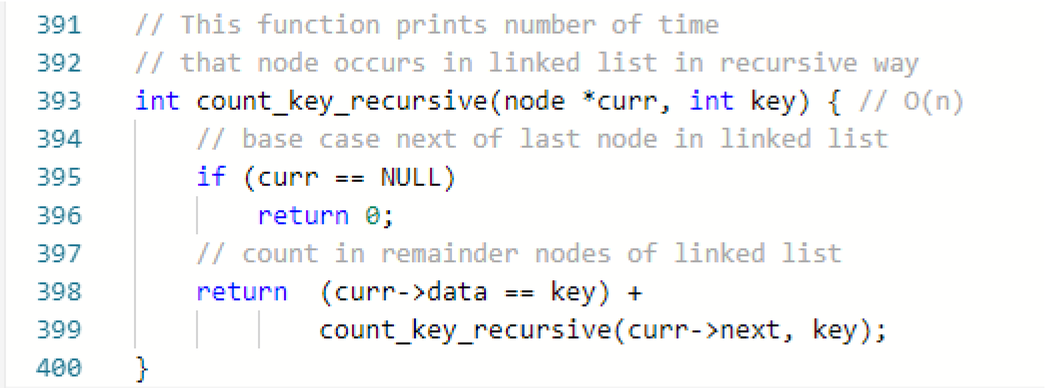
Count number of times which element occurs in linked list by recursive way

- Implement function which count number of times which element occurs in linked list in recursive way
- Function Name: count key recursion
- Parameters: pointer of node to head of linked list => (node* curr)

  int key which represent element
- Return: int number of occurs in linked list

Count number of times which element occurs in linked list by recursive way

Link: repl.it/repls/SmartSecondaryLanservers



```
// This function prints number of time
// that node occurs in linked list in recursive way
int count_key_recursive(node *curr, int key) { // O(n)
    // base case next of last node in linked list
    if (curr == NULL)
        return 0;
    // count in remainder nodes of linked list
    return  (curr->data == key) +
            count_key_recursive(curr->next, key);
}
```

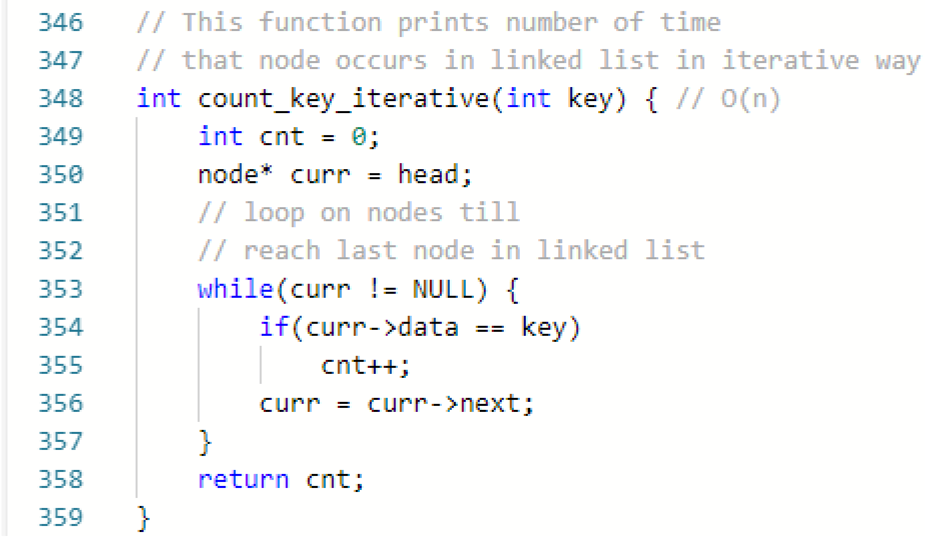
Count number of times which element occurs in linked list by iterative way

- Implement function which count number of times which element occurs in linked list in iterative way
- Function Name: count key iterative
- Parameters: pointer of node to head of linked list => (node* curr)

  int key which represent element
- Return: int number of occurs in linked list

Count number of times which element occurs in linked list by iterative way

Link: repl.it/repls/SmartSecondaryLanservers



```cpp
// This function prints number of time
// that node occurs in linked list in iterative way
int count_key_iterative(int key) { // O(n)
    int cnt = 0;
    node* curr = head;
    // loop on nodes till
    // reach last node in linked list
    while(curr != NULL) {
        if(curr->data == key)
            cnt++;
        curr = curr->next;
    }
    return cnt;
}
```

Practice

21- ~~Intersection of two sorted linked list~~

22- ~~Intersection of two unsorted linked list~~

23- ~~Union of two sorted linked list~~

24- ~~Union of two unsorted linked list~~

25- ~~Difference of two sorted linked list~~

26- ~~Difference of two unsorted linked list~~

27- ~~Segregate even and odd nodes in a linked list~~

28- ~~Check if linked list is palindrome or not~~

29- ~~Count number of times which element occurs in linked list by recursive way~~

30- ~~Count number of times which element occurs in linked list by iterative way~~

DO
MORE.

Assignment

References

[01]	Online Course YouTube Playlists	https://bit.ly/2Pq88rN
[02]	Introduction to Algorithms Thomas H. Cormen	https://bit.ly/2ONhuSn
[03]	Competitive Programming 3 Steven Halim	https://nus.edu/2z40vyK
[04]	Fundamental of Algorithmics Gilles Brassard and Paul Bartley	https://bit.ly/2QuvwbM
[05]	Analysis of Algorithms An Active Learning Approach	http://bit.ly/2EgCCYX
[06]	Data Structures and Algorithms Annotated Reference	http://bit.ly/2c37XEv
[07]	Competitive Programmer's Handbook	https://bit.ly/2APAbeG
[08]	GeeksforGeeks	https://geeksforgeeks.org
[09]	Codeforces Online Judge	http://codeforces.com
[10]	HackerEarth Online Judge	https://hackerearth.com
[11]	TopCoder Online Judge	https://topcoder.com

Questions ?

© Prepared by: Mohamed Ayman