**Single-linked list example:**

package trial2;

import java.util.Scanner;

/\*\*

\*

\* @author malavikah

\*/

class Node

{

int data;

Node link;

Node()

{

this.link=null;

}

}

public class Trial2 {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

Node s = new Node();

Node t=s;

System.out.println("Enter first element ");

s.data=sc.nextInt();

while (true){

System.out.println("Do you want to add an element to LL: enter y or n");

char c= sc.next().charAt(0);

if (c == 'n' || c == 'N')

{

break;

}

s.link=new Node();

System.out.println("Enter the element to enter");

s.link.data=sc.nextInt();

s=s.link;

}

s=t;

while(s!=null)

{

System.out.print(s.data+" -> ");

s=s.link;

}

System.out.print("null");

}

}

* **Write code to remove duplicates from an unsorted linked list.**

package trial2;

import java.util.Scanner;

/\*\*

\*

\* @author malavikah

\*/

class Node

{

int data;

Node link;

Node()

{

this.link=null;

}

}

public class Trial2 {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

Node s = new Node();

Node t=s;

System.out.println("Enter first element ");

s.data=sc.nextInt();

while (true){

System.out.println("Do you want to add an element to LL: enter y or n");

char c= sc.next().charAt(0);

if (c == 'n' || c == 'N')

{

break;

}

s.link=new Node();

System.out.println("Enter the element to enter");

s.link.data=sc.nextInt();

s=s.link;

}

s=t;

while(s!=null)

{

System.out.print(s.data+" -> ");

s=s.link;

}

System.out.print("null");

s=t;

Node start=s;

Node i=new Node();

// traverse through each node and compare that node with every other node

while(s!=null)

{

//make t as 's' pointer so you always can track it back

t=s;

//i should alwasy point to the next node as s

i=s.link;

//traverse thru other nodes to compare with 's'

while(i!=null)

{

//node when it is equal

if(i.data==s.data)

{

// to traverse to find the node behind the detected node so you can skip the repeated elements

while(s!=null)

{

//when the previous node has been detected

if(s.link==i)

{

//remove the link of the repeated node and give it to the next node.

s.link=i.link;

}

s=s.link;

}

System.out.println("here");

}

i=i.link;

s=t;

}

//move 's' accrosee the list of nodes to compare with every other node.

s=s.link;

}

s=start;

while(s!=null)

{

System.out.print(s.data+" -> ");

s=s.link;

}

System.out.print("null");

}

}

* **Implement an algorithm to find the nth to last element of a singly linked list.**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package trial2;

import java.util.Scanner;

/\*\*

\*

\* @author malavikah

\*/

class Node

{

int data;

Node link;

Node()

{

this.link=null;

}

}

public class Trial2 {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

Node s = new Node();

Node t=s;

System.out.println("Enter first element ");

s.data=sc.nextInt();

while (true){

System.out.println("Do you want to add an element to LL: enter y or n");

char c= sc.next().charAt(0);

if (c == 'n' || c == 'N')

{

break;

}

s.link=new Node();

System.out.println("Enter the element to enter");

s.link.data=sc.nextInt();

s=s.link;

}

s=t;

while(s!=null)

{

System.out.print(s.data+" -> ");

s=s.link;

}

System.out.print("null");

s=t;

Node l=new Node();

int count=0;

l=s;

System.out.println("Enter the nth place from last to display");

int val=sc.nextInt();

while(s!=null)

{

//keep track of how many nodes are present in the linked list.

count++;

if(s.link==null)

{

// to trverse through the linked list to find that particular element.

//can use while loop also

for(int i=0;i<count;i++)

{

l=l.link;

// when it matches the nth element from the end.

if(i==(count-val-1))

{

// print the data

System.out.println(l.data);

}

}

}

s=s.link;

}

}

}

* **Implement an algorithm to delete a node in the middle of a single linked list, given only access to that node.  EXAMPLE  Input: the node ‘c’ from the linked list a->b->c->d->e  Result: nothing is returned, but the new linked list looks like a->b->d->e**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package trial2;

import java.util.Scanner;

/\*\*

\*

\* @author malavikah

\*/

class Node

{

int data;

Node link;

Node()

{

this.link=null;

}

}

public class Trial2 {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

Node s = new Node();

Node t=s;

System.out.println("Enter first element ");

s.data=sc.nextInt();

while (true){

System.out.println("Do you want to add an element to LL: enter y or n");

char c= sc.next().charAt(0);

if (c == 'n' || c == 'N')

{

break;

}

s.link=new Node();

System.out.println("Enter the element to enter");

s.link.data=sc.nextInt();

s=s.link;

}

s=t;

while(s!=null)

{

System.out.print(s.data+" -> ");

s=s.link;

}

System.out.print("null");

s=t;

Node l=new Node();

int count=0;

l=s;

System.out.println("Enter the element in the linked list to delete");

int val=sc.nextInt();

while(s!=null)

{

if(s.data==val)

{

// Copy the next node's data into this present one and delete the next node

// Never found it to be this simple

s.data=s.link.data;

s.link=s.link.link;

}

s=s.link;

}

s=t;

while(s!=null)

{

System.out.print(s.data+" -> ");

s=s.link;

}

System.out.print("null");

}

}

* **MERGE TWO SORTED LINKED LIST TO A SINGLE ONE**
* **struct** node\* SortedMerge(**struct** node\* a, **struct** node\* b)
* {
* **struct** node\* result = NULL;
* /\* Base cases \*/
* **if** (a == NULL)
* **return**(b);
* **else** **if** (b==NULL)
* **return**(a);
* /\* Pick either a or b, and recur \*/
* **if** (a->data <= b->data)
* {
* result = a;
* result->next = SortedMerge(a->next, b);
* }
* **else**
* {
* result = b;
* result->next = SortedMerge(a, b->next);
* }
* **return**(result);
* }
* **DETECT LOOP IN LINKED LIST**
* **void** removeLoop(**struct** node \*, **struct** node \*);
* /\* This function detects and removes loop in the list
* If loop was there in the list then it returns 1,
* otherwise returns 0 \*/
* **int** detectAndRemoveLoop(**struct** node \*list)
* {
* **struct** node  \*slow\_p = list, \*fast\_p = list;
* **while** (slow\_p && fast\_p && fast\_p->next)
* {
* slow\_p = slow\_p->next;
* fast\_p  = fast\_p->next->next;
* /\* If slow\_p and fast\_p meet at some point then there
* is a loop \*/
* **if** (slow\_p == fast\_p)
* {
* removeLoop(slow\_p, list);
* /\* Return 1 to indicate that loop is found \*/
* **return** 1;
* }
* }
* /\* Return 0 to indeciate that ther is no loop\*/
* **return** 0;
* }

ADD TWO LINKED LIST

**struct** node\* addTwoLists (**struct** node\* first, **struct** node\* second)

{

**struct** node\* res = NULL; // res is head node of the resultant list

**struct** node \*temp, \*prev = NULL;

**int** carry = 0, sum;

**while** (first != NULL || second != NULL) //while both lists exist

    {

        // Calculate value of next digit in resultant list.

        // The next digit is sum of following things

        // (i)  Carry

        // (ii) Next digit of first list (if there is a next digit)

        // (ii) Next digit of second list (if there is a next digit)

        sum = carry + (first? first->data: 0) + (second? second->data: 0);

        // update carry for next calulation

        carry = (sum >= 10)? 1 : 0;

        // update sum if it is greater than 10

        sum = sum % 10;

        // Create a new node with sum as data

        temp = newNode(sum);

        // if this is the first node then set it as head of the resultant list

**if**(res == NULL)

            res = temp;

**else** // If this is not the first node then connect it to the rest.

            prev->next = temp;

        // Set prev for next insertion

        prev  = temp;

        // Move first and second pointers to next nodes

**if** (first) first = first->next;

**if** (second) second = second->next;

    }

**if** (carry > 0)

      temp->next = newNode(carry);

    // return head of the resultant list

**return** res;

}

SPLIT LINKED LIST IN BETWEEN AND REVERSE THE SECOND HALF AND MERGE IT BACK

Node\* Reverse(Node \*head)

{

struct Node\* slow = head;

struct Node\* fast = head;

struct Node\* current = head;

struct Node\* newlist = NULL;

struct Node\* first = NULL;

while (fast !=NULL)

{

current = slow;

slow=slow->next;

fast = (fast->next) ? fast->next->next : NULL;

}

current->next = NULL;

first = head;

newlist = slow;

Node \*temp= first;

struct Node\* prev = NULL;

struct Node\* curr = newlist;

struct Node\* next;

while (curr != NULL)

{

next = curr->next;

curr->next = prev;

prev = curr;

curr = next;

}

newlist = prev;

while(first->next!=NULL)

{

first=first->next;

}

first->next = newlist;

first=temp;

return first;

}

Given a linked list, reverse alternate nodes and append at the end

Examples

Input List: 1->2->3->4->5->6

Output List: 1->3->5->6->4->2

Input List: 12->14->16->18->20

Output List: 12->16->20->18->14

The idea is to maintain two linked lists, one list of all odd positioned nodes (1, 3, 5 in above example) and other list of all even positioned nodes (6, 4 and 2 in above example). Following are detailed steps.

**1)** Traverse the given linked list which is considered as odd list. Do following for every visited node.

……**a)** If the node is even node, remove it from odd list and add it to the front of even node list. Nodes are added at front to keep the reverse order.

**2)** Append the even node list at the end of odd node list.

#include<stdio.h>

#include<stdlib.h>

/\* A linked list node \*/

**struct** node

{

**int** data;

**struct** node \*next;

};

/\* Function to reverse all even positioned node and append at the end

   odd is the head node of given linked list \*/

**void** rearrange(**struct** node \*odd)

{

    // If linked list has less than 3 nodes, no change is required

**if** (odd == NULL || odd->next == NULL || odd->next->next == NULL)

**return**;

    // even points to the beginning of even list

**struct** node \*even = odd->next;

    // Remove the first even node

    odd->next = odd->next->next;

    // odd points to next node in odd list

    odd = odd->next;

    // Set terminator for even list

    even->next = NULL;

    // Traverse the  list

**while** (odd && odd->next)

    {

       // Store the next node in odd list

**struct** node \*temp = odd->next->next;

       // Link the next even node at the beginning of even list

       odd->next->next = even;

       even = odd->next;

       // Remove the even node from middle

       odd->next = temp;

       // Move odd to the next odd node

**if** (temp != NULL)

         odd = temp;

    }

    // Append the even list at the end of odd list

    odd->next = even;

}

/\* Function to add a node at the beginning of Linked List \*/

**void** push(**struct** node\*\* head\_ref, **int** new\_data)

{

**struct** node\* new\_node = (**struct** node\*) **malloc**(**sizeof**(**struct** node));

    new\_node->data  = new\_data;

    new\_node->next = (\*head\_ref);

    (\*head\_ref)    = new\_node;

}

/\* Function to print nodes in a given linked list \*/

**void** printList(**struct** node \*node)

{

**while** (node != NULL)

    {

**printf**("%d ", node->data);

        node = node->next;

    }

}

/\* Druver program to test above function \*/

**int** main()

{

**struct** node \*start = NULL;

    /\* The constructed linked list is:

     1->2->3->4->5->6->7 \*/

    push(&start, 7);

    push(&start, 6);

    push(&start, 5);

    push(&start, 4);

    push(&start, 3);

    push(&start, 2);

    push(&start, 1);

**printf**("\n Linked list before calling  rearrange() ");

    printList(start);

    rearrange(start);

**printf**("\n Linked list after calling  rearrange() ");

    printList(start);

**return** 0;

}

REMOVE DUPLICATES FROM SORTED LINKED LIST;

#include <iostream>

using namespace std;

struct node

{

int data;

struct node \*next;

};

void add(struct node \*\*head,int data)

{

struct node\* new\_node = (struct node\*)malloc(sizeof(struct node));

new\_node->data = data;

new\_node->next = \*head;

\*head = new\_node;

}

void printList(struct node \*node)

{

while(node!=NULL)

{

printf("%d ", node->data);

node = node->next;

}

}

void remove(struct node\* head)

{

struct node\* curr = head;

while(curr->next!=NULL)

{

if( curr->data == curr->next->data)

{

struct node\* new1 = NULL;

new1= curr->next->next;

free(curr->next);

curr->next = new1;

}

else

{

curr=curr->next;

}

}

}

int main() {

struct node\* head = NULL;

add(&head, 20);

add(&head, 13);

add(&head, 13);

add(&head, 11);

add(&head, 11);

add(&head, 11);

printf("\n Linked list before duplicate removal ");

printList(head);

/\* Remove duplicates from linked list \*/

remove(head);

printf("\n Linked list after duplicate removal ");

printList(head);

return 0;

}