Hackerrank id : https://www.hackerrank.com/jubayer\_1996Hus1

(@jubayer\_1996Hus1)

1. Print the Elements of a Linked List :

Answer :

void printLinkedList(SinglyLinkedListNode\* head) {

SinglyLinkedListNode\* temp = head;

while(temp!=NULL)

{

cout<<temp->data<<"\n";

temp=temp->next;

}}

2.Insert a Node at the Tail of a Linked List :

Answer :

SinglyLinkedListNode\* insertNodeAtTail(SinglyLinkedListNode\* head, int

data) {

SinglyLinkedListNode\* node = new SinglyLinkedListNode(data);

node->data =data ;

node->next=NULL;

if(head==NULL)

{

head = node ;

}

else

{

node->data=data ;

SinglyLinkedListNode\*n ;

n = head;

while(n->next!=NULL){

n=n->next;

}

n->next=node;

}

return head ;

}

3.Insert a node at the head of a linked list :

Answer :

SinglyLinkedListNode\* insertNodeAtHead(SinglyLinkedListNode\* head, int

data) {

SinglyLinkedListNode\* node = new SinglyLinkedListNode(data);

node-> data =data ;

node-> next=NULL;

if(head==NULL)

{

head = node ;

}

else

{

node->next=head ;

head = node ;

}

return head ;

}

4.Insert a node at a specific position in a linked list:

Answer:

SinglyLinkedListNode\* insertNodeAtPosition(SinglyLinkedListNode\* head,

int data, int position) {

SinglyLinkedListNode\* node = new SinglyLinkedListNode(data);

int i ;

node->data =data ;

node->next=NULL;

SinglyLinkedListNode\* n ;

n=head;

while(i<position-1)

{

n=n->next;

i++;

}

node->next=n->next;

n->next=node;

return head ;

}

5.Delete a Node

Answer :

SinglyLinkedListNode\* deleteNode(SinglyLinkedListNode\* head, int

position) {

if(position == 0)

{

return head->next;

}

else

{

SinglyLinkedListNode\* a = head;

for(int i = 1; i < position; i++){

a = a->next;

}

a->next = a->next->next;

return head;

}} 6.Print in Reverse :

Answer :

void reversePrint(SinglyLinkedListNode\* head) {

if(!head)

return;

reversePrint(head->next);

cout<<head->data<<endl;

}

7.Reverse a linked list :

Answer :

SinglyLinkedListNode\* reverse(SinglyLinkedListNode\* head) {

SinglyLinkedListNode\* temp = head;

if(head->next)

{

temp=reverse(head->next);

head->next->next=head;

head->next=NULL;

}

return temp;

}

8.Compare two linked lists :

Answer :

bool compare\_lists(SinglyLinkedListNode\* head1, SinglyLinkedListNode\*

head2) {

SinglyLinkedListNode\* n1 = head1;

SinglyLinkedListNode\* n2 = head2;

while (n1||n2)

{

if (!n1||!n2)

return 0;

if (n1->data!=n2->data)

return 0;

n1=n1->next;

n2 = n2->next;

}

return 1;

}

9.Merge two sorted linked lists:

Answer :

SinglyLinkedListNode\* mergeLists(SinglyLinkedListNode\* head1,

SinglyLinkedListNode\* head2) {

SinglyLinkedListNode\* x;

SinglyLinkedListNode\* y;

SinglyLinkedListNode\* z;

SinglyLinkedListNode\* head =NULL;

x= head1;

y=head2;

if(x->data<=y->data)

{

z=x;

x=z->next;

}

else

{

z=y;

y=z->next;

}

head=z;

while(x&&y){

if(x->data<=y->data)

{

z->next=x;

z=x;

x=z->next;

}

else

{

z->next=y;

z=y;

y=z->next;

}

}

if(x==NULL)

{

z->next=y;

}

else

{

z->next= x;

}

return head ;

}

10.Get Node Value :

Answer :

int getNode(SinglyLinkedListNode\* head, int positionFromTail) {

if(head==nullptr)

return -1 ;

int i=0 , j =0 ;

SinglyLinkedListNode\* x = head ;

SinglyLinkedListNode\* y = head ;

while (x->next!=NULL)

{

j++;

x=x->next;

if(j-i>positionFromTail)

{

y = y->next;

i++;

}

}

return y->data ;

}

11.Delete duplicate-value nodes from a sorted linked list :

Answer :

SinglyLinkedListNode\* removeDuplicates(SinglyLinkedListNode\* head) {

if(!head)

return head ;

SinglyLinkedListNode\* node = head->next;

SinglyLinkedListNode\* temp = head ;

while(node!=NULL)

{

if(node->data==temp->data)

{

SinglyLinkedListNode\* node1 = node ;

node= node->next;

temp->next=node;

free(node1);

}

else

{

temp=node;

node=node->next;

}

}

return head ;

}

12.Cycle Detection :

Answer :

bool has\_cycle(SinglyLinkedListNode\* head) {

if(head==nullptr)

return 0;

SinglyLinkedListNode\* node = head ;

SinglyLinkedListNode\* temp =head ;

while (temp !=nullptr && temp->next != nullptr )

{

temp = temp->next->next;

node = node->next;

if(temp==node)

return 1 ;

}

return 0 ;

}

13.Find Merge Point of Two Lists :

Answer :

int findMergeNode(SinglyLinkedListNode\* head1, SinglyLinkedListNode\*

head2) {

SinglyLinkedListNode\* node;

SinglyLinkedListNode\* temp;

node=head1;

temp=head2;

while(node!=temp)

{

if(node->next==NULL)

{

node = head2;

}

else

{

node = node->next;

}

if(temp->next==NULL)

{

temp = head1;

}

else

{

temp = temp->next;

}

}

return node->data;

}

14.Inserting a Node Into a Sorted Doubly Linked List :

Answer :

DoublyLinkedListNode\* sortedInsert(DoublyLinkedListNode\* head, int

data) {

if(!head)

return head ;

DoublyLinkedListNode\* node = new DoublyLinkedListNode(data);

if (head->data>= node->data)

{

head->prev = node;

node->next = head;

return node;

}

DoublyLinkedListNode\* node1 = head;

while(node1!=NULL)

{

if(node1->data >= node->data)

{

node1->prev->next = node ;

node->prev= node1->prev ;

node1->prev= node ;

node->next= node1 ;

break ;

}

else if (node1->next ==NULL )

{

node1-> next = node;

node->prev = node1 ;

break ;

}

node1 = node1->next;

}

return head ;

}

15. Reverse a doubly linked list :

Answer :

DoublyLinkedListNode\* reverse(DoublyLinkedListNode\* head) {

DoublyLinkedListNode\* temp = NULL;

DoublyLinkedListNode\* current = head;

while (current != NULL)

{

temp = current->prev;

current->prev = current->next;

current->next = temp;

current = current->prev;

}

if(temp != NULL )

{

head = temp->prev;

}

return head ;

}