Practice Set: Linked List

For all the questions assume following struct is defined:-

```
struct node {
    int value;
    struct node *next;
};


typedef struct node Node;
```

MSQ Question 1

Consider the following pseudo C program:

```
struct point {
        int X, Y;
};
typedef struct point P;


struct ptlist {
        P pt;
        struct ptlist *next;
};
struct ptlist *head;
```

Which method should be used to access the member X using the pointer head ?

```
(1) head → pt → X
(2) head → pt.X
(3) (*head).pt.X
(4) (*head).pt → X
```

Answer: B,C

Question 2

Consider a function given below and mark correct option for the linked list initially having values 1,6, 9,13

```
int find(struct node *head, int val){
    if (head==NULL) return 0;

    while(head->next!=NULL){
        if (head->value == val)
        return 1;
        head=head->next;
    }
    return 0;

}
```

A. Function returns 1 for 6 i.e. `find(head, 6)` is 1
B. Function returns 1 for 13 i.e. `find(head, 13)` is 1
C. Function returns 1 if value exists in linked list
D. Function returns 1 if value does not exist in linked list

Answer A

MSQ Question 3

Consider the given C-program:

```
struct node
{
    int data;
    struct node * next;
};
struct node *ptr;
```

Which of the following represents the correct way to create a new node?

```
(1)    ptr = (struct node*) malloc (sizeof(struct  node));
(2)    ptr = (struct node*) malloc (sizeof( ptr ));
(3)    ptr = (struct node*) malloc (sizeof(struct  node*));
(4)    ptr = (struct node*) malloc (sizeof(*ptr) );
```

Answer B,D

Question 4

Which of the following functions correctly inserts a value x at the front of a linked list and returns a reference to the new front of the linked list ?

```
I .
Node* insertFront(Node* first, int x) {
 first = malloc(sizeof( struct node  ));
    first->value = x;
    first->next = first;
 return first;
}
```

```
II .
Node* insertFront(Node* first, int x) {
 Node* newNode = malloc(sizeof( struct node  ));
    newNode->value = x;
    newNode->next = first;
 return newNode;
}
```

```
III.
Node* insertFront(Node* first, int x) {
Node* newNode = malloc(sizeof( struct node  ));
    first =newNode;
    newNode->value = x;
    newNode->next = first;
return newNode;
}
```

Answer II

Question 5

```
Node* recursive_remove(Node* head, int x)
{
    if (head == NULL)
        return NULL;

    if (head->value == x)
    {
        Node* tmp = head->next;
        free(head);
        return tmp;
    }
    else
    {
        head->next = recursive_remove(head->next, x);
        return head;
    }
}
```

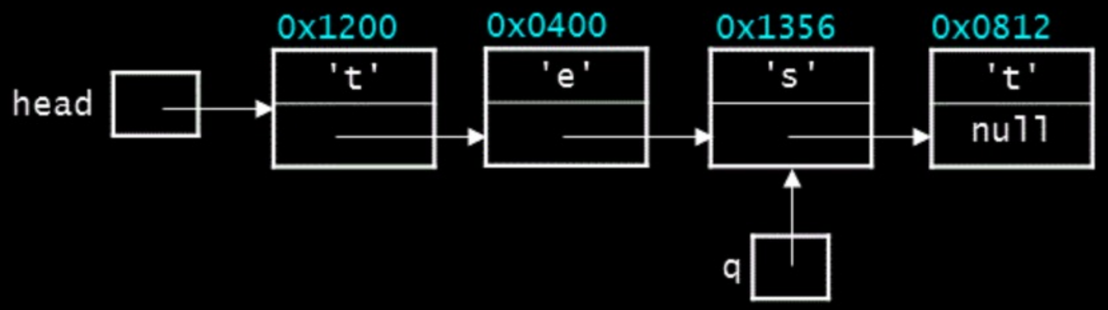
If the function recursive_remove(head, 2) is applied to a linked list initially having values 1, 2, 2, 8, 6, 2, 2, which of the following options correctly represents the final linked list after the operation?

A. Final LinkedList will be 1,2,8,6,2,2
B. Final LinkedList will be 1,8,6
C. Final LinkedList will be 1,8,6,2,2
D. None of the above

Answer A

Question 6

The diagram below shows a linked list of characters, along with two variables that each store a reference/pointer to one of the nodes in the list



Colored numbers are memory addresses of corresponding nodes.

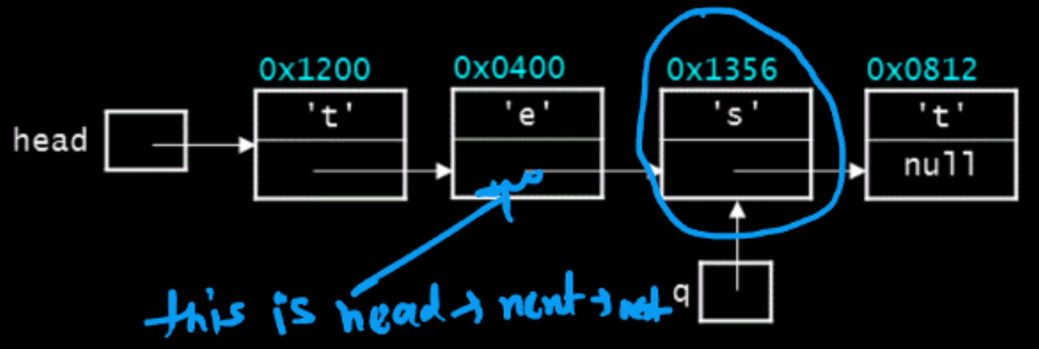
a) On the diagram, circle the memory location specified by the expression head->next->next

b) What is the value of the expression from part a?

c) Modify the diagram above to reflect the results of executing the following lines on the original version of the list

- q = q->next;
- q->next = head;

Answer



a. On the diagram, circle the memory location specified by the expression
head->next->next

b. What is the value of the expression from part a? 0x1356

c. Modify the diagram above to reflect the results of executing the following lines on the original version of the list

```
q = q->next;
q->next = head;
```

} ← this will make circular LL.

Question 7

Consider the function below which indent to remove last node of the linked list.

```
void removeLast(Node* first) {
    Node *p, *q;
    p = first;
    q = p->next;
    while (q->next != NULL) {
    p = q;
    q = q->next;
    }
    p->next = NULL;
    free(q);
}
```

Which class of linked lists does the provided function correctly work for, given that the parameter 'first' refers to the first node in the linked list (or null if it doesn't exist) ?

A. No linked lists (The function does not remove last node of any given linked list)

B. All nonempty linked lists

C. All linked lists with more than one node

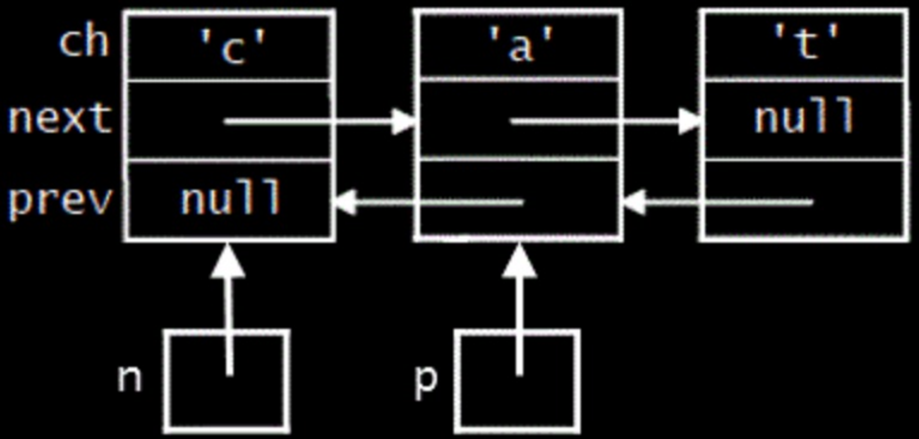
D. Empty linked lists and all linked lists with more than one node

E. All linked lists

Answer C

Question 8

Consider the doubly linked list below



```
struct node {
    char ch;
    struct node *prev;
    struct node *next;
};
```

Which of the following expressions does not refer to the third node in the list?

1.p->next
2.n->next->next

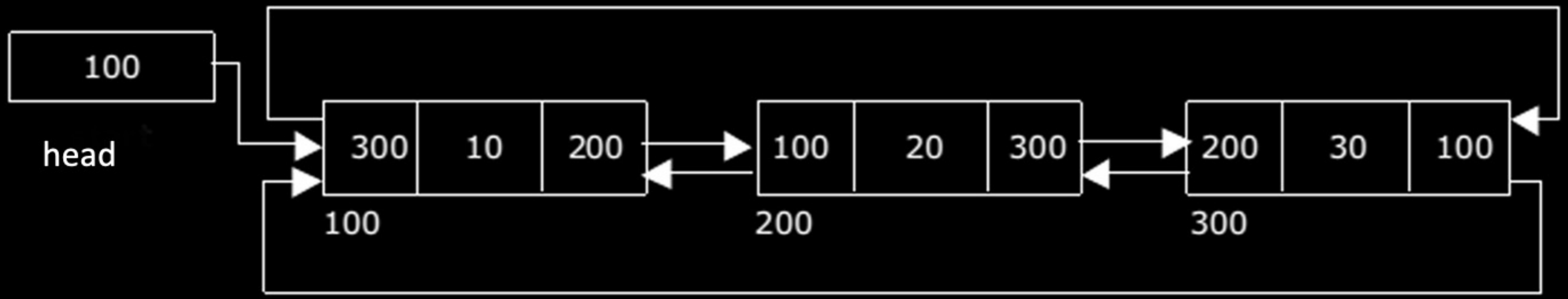
3. p->prev->next
4. p->prev->next->next

5. n->next->next->prev->next

Answer 3

Question 9

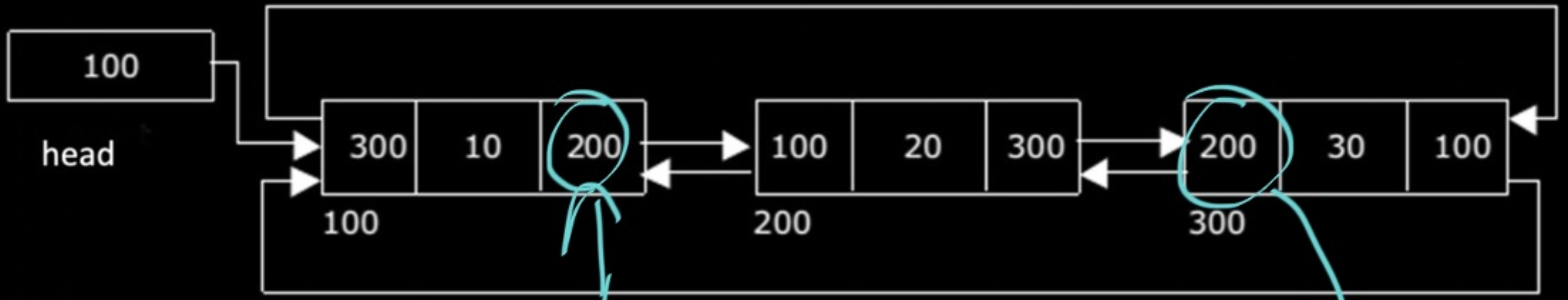
Consider the following “Circular Doubly linked list”.



Suppose we want to delete a node with address 200 then how many links we need to change ?

Answer

Consider the following “Circular Doubly linked list”.



it should get value 300

it shoud got valuo 100

Suppose we want to delete a node with address 200 then how many links we need to change ?

```
struct node *tmp = head;
temp = head->next->next
temp->prev = head;
free(head->next);
head->next=temp;
```

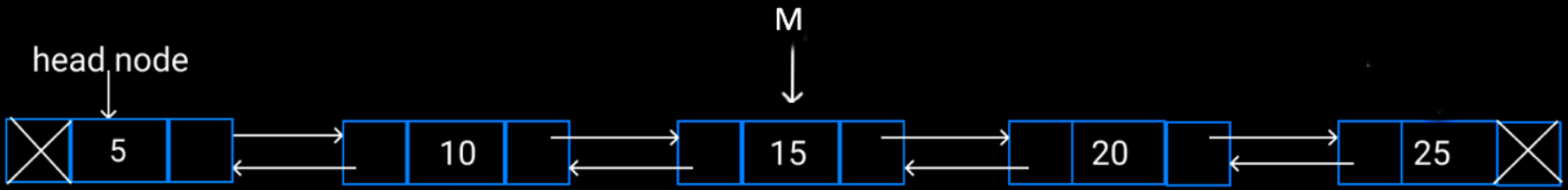
(temp = 300)

(Node with address 300 will get 100 in its Prev field)

free node with address 200

2 links answer

Question 10



Consider the above linked list and perform following operations.
Take each set of operations as separate from other (i.e. A,B and C are separate)
Draw final diagram after each A, B and C.

A.

```
head = head->next;
head->prev = null;
```

B.

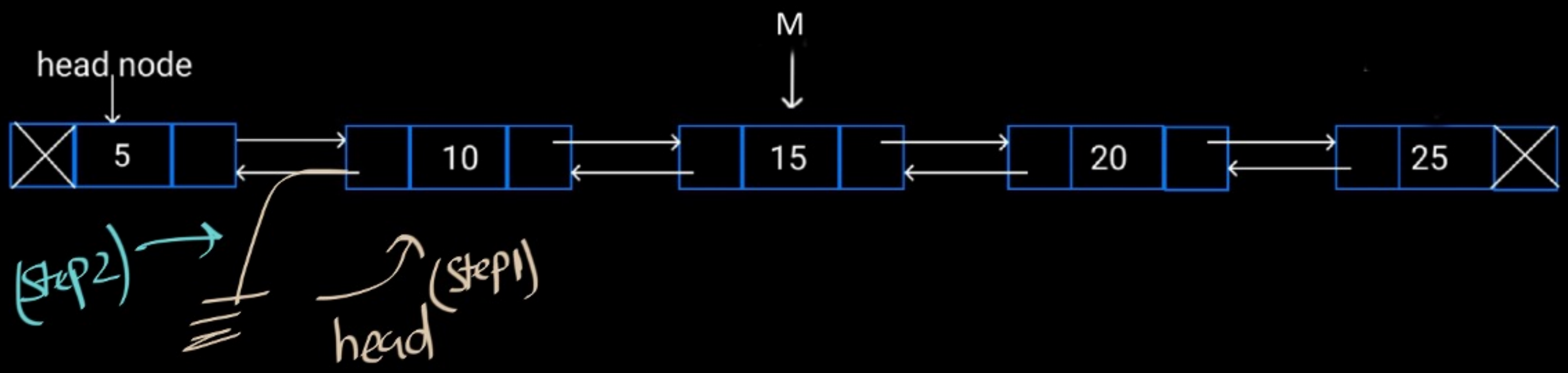
```
struct node *P = malloc(sizeof(struct node));
//this is doubly linked list node
P->next = M->next;
M->next = P;
P-next->prev = P;
p->prev = M;
```

C. On next page

C.

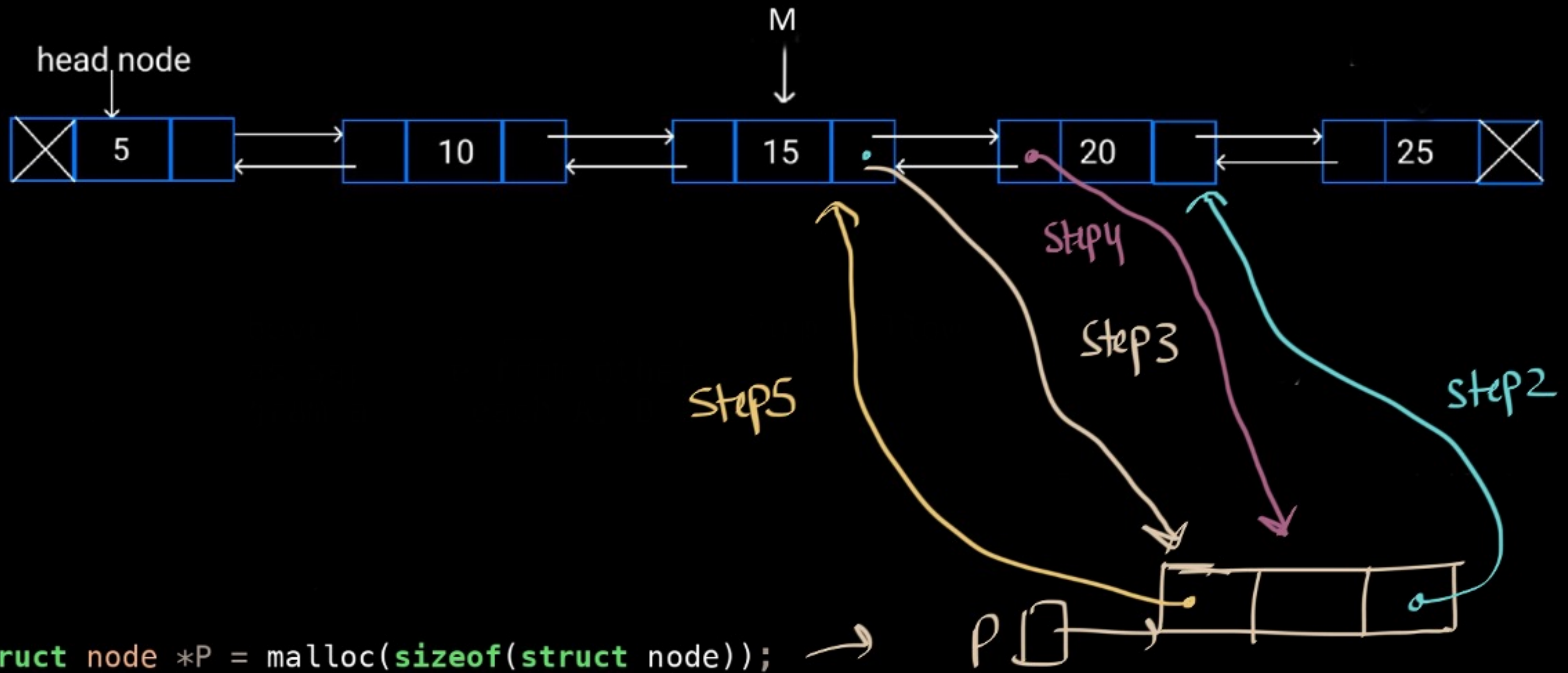
```
struct node *P, *Q, *N; //this is doubly linked list node
Q = head->next;
P = M->prev;
N = M->next;
P->next = head;
head->prev = P;
head->next = N;
N->prev = head;
M->next = Q;
Q->prev=M;
M->prev=NULL;
head=M;
```

Solution for A



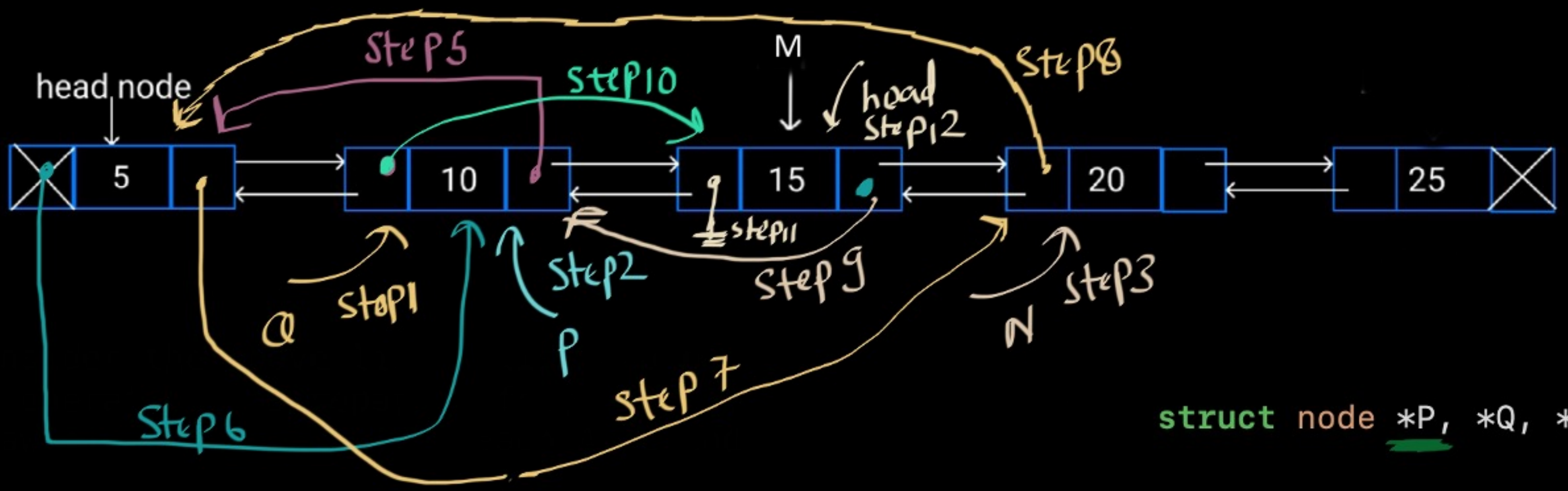
```
A.
head = head->next;
head->prev = null;
```

Solution for B

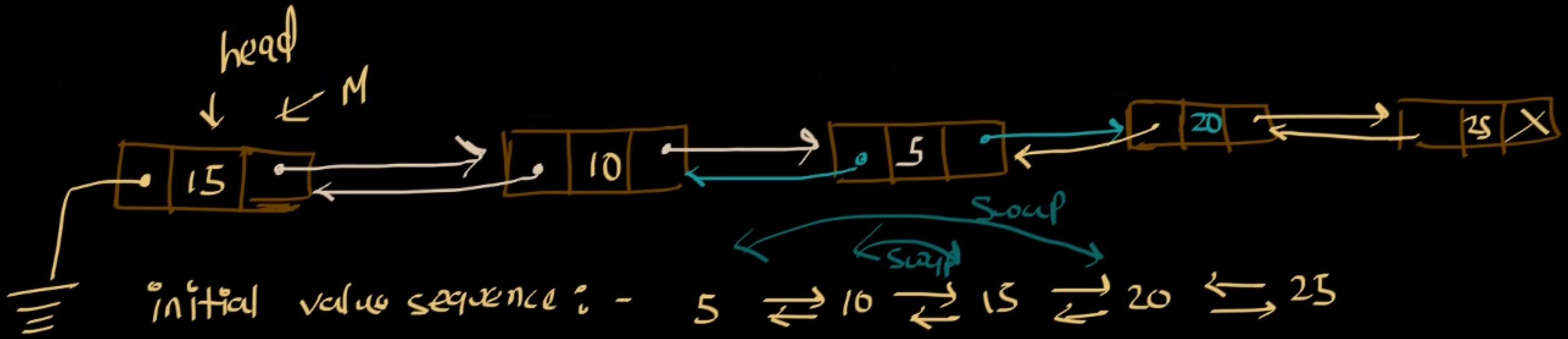


```
B.
struct node *P = malloc(sizeof(struct node));
//this is doubly linked list node
P->next = M->next;
M->next = P;
P-next->prev = P;
p->prev = M;
```

Solution for C



```
struct node *P, *Q, *N;
```

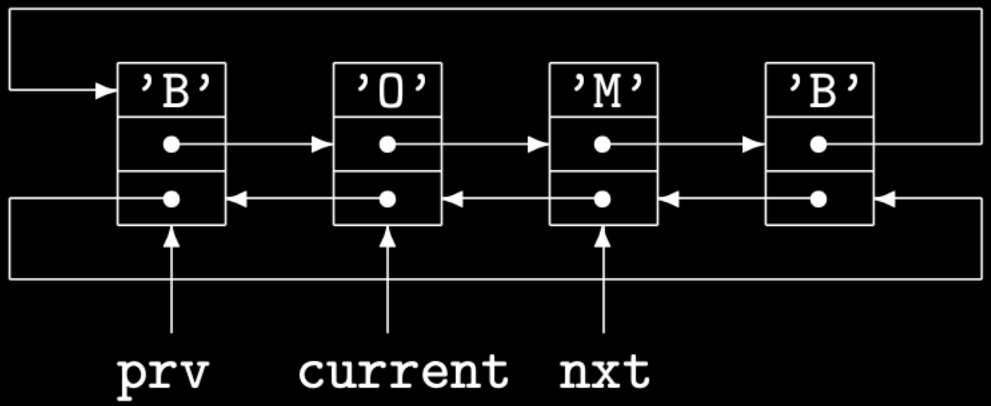


initial value sequence :- 5 ⇄ 10 ⇄ 15 ⇄ 20 ⇄ 25

final value sequence 15 ⇄ 10 ⇄ 5 ⇄ 20 ⇄ 25

Question 11

Consider Circular linked list given below

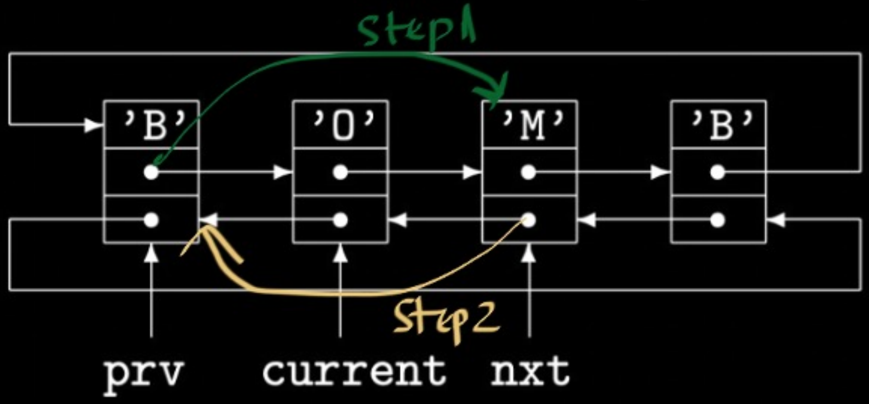


```
struct Node
{
char data;
struct Node *next;
struct Node *prev;
};
```

What should be final list after performing below code ?

```
prv->next = current->next;
nxt->prev= current->prev;
free(current);
```

Consider Circular linked list given below



```
struct Node
{
char data;
struct Node *next;
struct Node *prev;
};
```

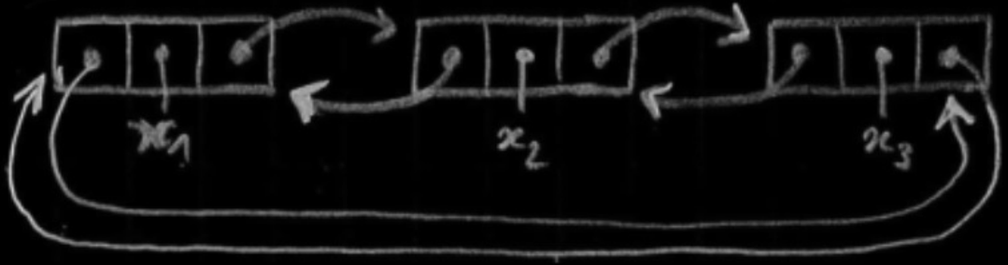
What should be final list after performing below code ?

```
prv->next = current->next;
nxt->prev= current->prev;
free(current);
```

Question 12

Given some nonempty **doubly linked circular list**, and current pointer points to some node in list.

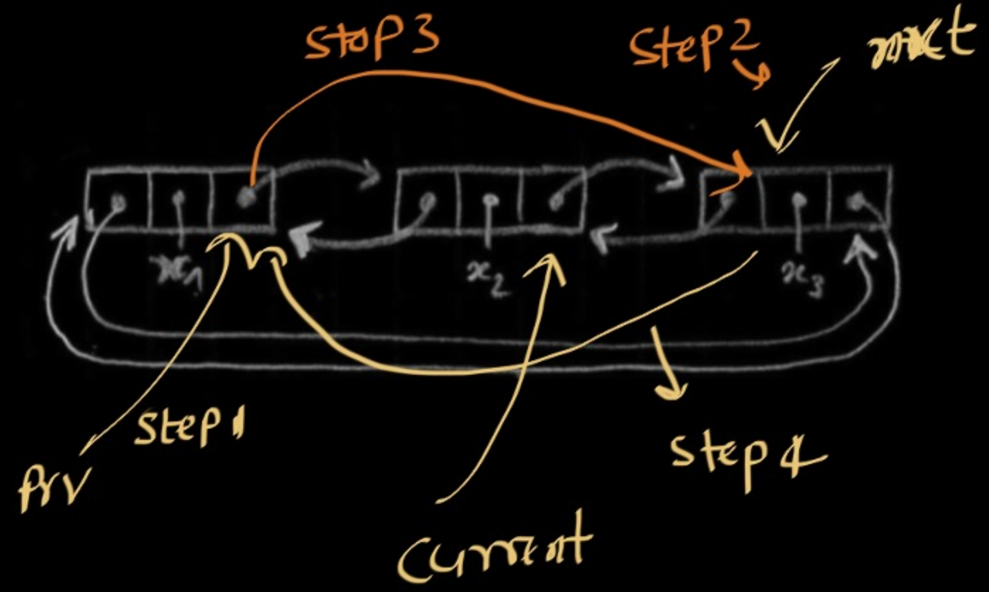
```
void mystry ( Node* current )
{
  Node* prv = current->prev;
  Node* nxt = current->next;
  prv->next = current->next;
  nxt->prev = current->prev;
  free(current);
}
```



```
A. Removes current node
B. Remove previous of current node
C. Remove next of current node
D. None of the above
```

Question:
Given some nonempty **doubly linked circular list**, and current pointer points to some node in list.

```
void mystry ( Node* current )
{
   Node* prv = current->prev;
   Node* nxt = current->next;
   prv->next = current->next;
   nxt->prev = current->prev;
   free(current);
}
```



[✓ A. Removes current node]
B. Remove previous of current node
C. Remove next of current node
D. None of the above

Question 13

Consider the following function CT which takes head pointer of two (singly) circular linked list.

```
struct node* CT(struct node *list1, struct node *list2)
{
    struct node *temp;
    if (list1==NULL)
        return list2;
    if (list2==NULL)
        return list1;
    tmp = list1->next;
    list1->next = list2->next;
    list2->next = tmp;
    return list2;
}
```

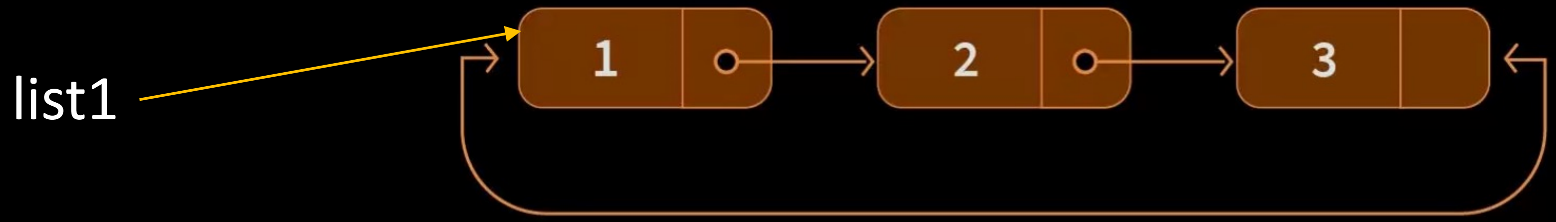
What Is this function doing ?

Answer

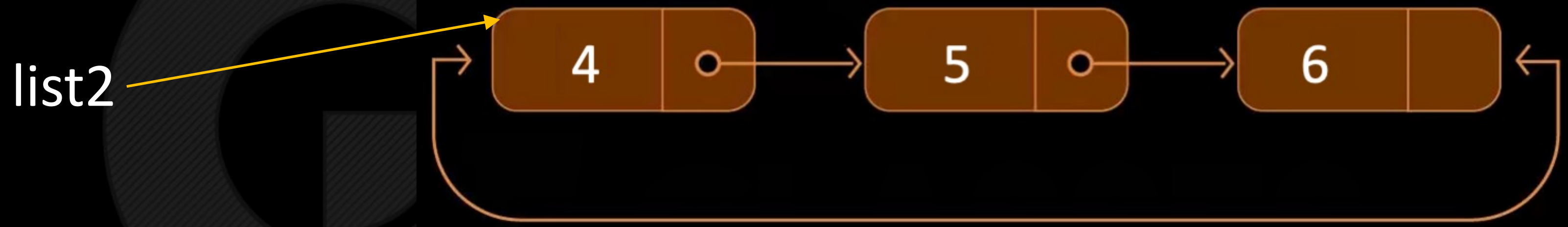
It appends two circular linked list in some way.

Example on the next page.

Data Structures

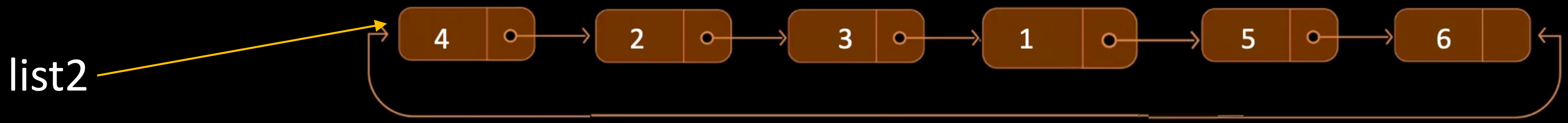


list1



list2

After the function call: -



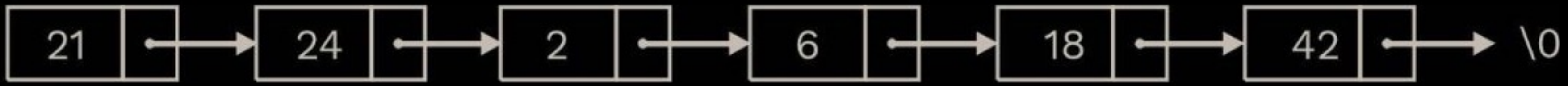
list2

Question 14

Consider the given C-code:

```
void func(struct node *head)
{  if (head == NULL)
   return;
   func(head → next);
   printf("%d", head → data);
}
```

If there exists a linked list:



Then what would be the last value printed by func?

Answer 21

Question 15

Consider the given C-code:

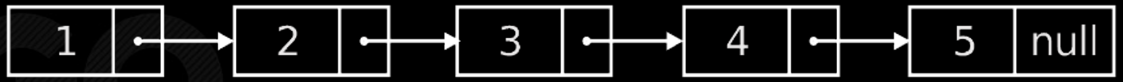
```
void func (struct node * head)
{
    if (head = = NULL)
        return;
printf("%d", head → data);

if (head → next ! = NULL)
    func(head → next → next);

printf ("%d"; head → data);

}
```

Given linked list:



What would be the output of the program?

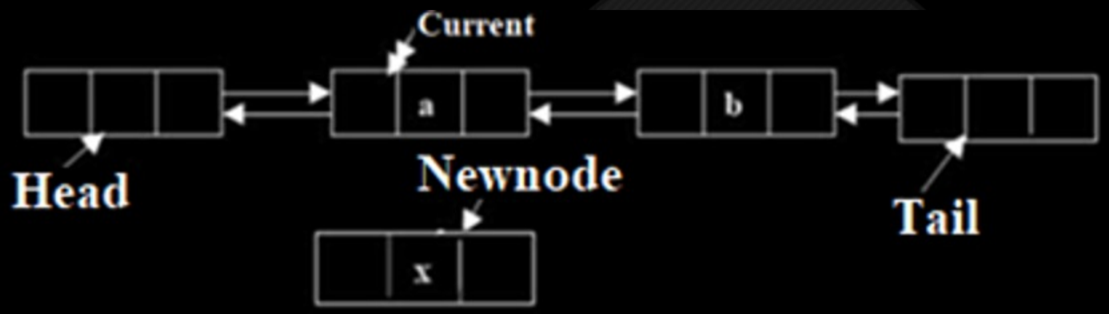
(1) 1 2 3 4 5
(2) 1 3 5 5 3 1
(3) 5 4 3 2 1
(4) 1 2 4 4 2 1

Solution

Ans. (2)
The function ‘func’ prints the data of linked lists from head to tail skipping even nodes and then prints data of linked list in reverse order skipping even nodes. Basically it prints odd nodes data as 1 3 5 5 3 1.

Question 16

Which of the following correctly describe(s) the steps in inserting elements x immediately after the element a in the above doubly linked list?



(1)

```
current→next=x;
x→prev=current;
x→next=b;
b→prev=x;
```

(2)

```
current→next=x;
x→prev=current→prev;
x→next=b;
b→prev=x;
```

(3)

```
a→next=x;
x→prev=current;
x→next=b;
b→prev=x;
```

(4)

```
Newnode→next= Current→next;
Newnode→prev=Current;
Newnode→prev→next=Newnode;
Newnode→next→prev=Newnode;
```

Solution

Option (4) Here when we are inserting the new node at the intermediate position in the doubly linked list, we have to change the four-pointers.
First, we must change the pointer of the new node.
Newnode.next= Current.next // new node right pointer starts pointing to the node with element b.
Newnode.prev=Current // new node left pointer starts pointing to the node with element a.
Newnode.prev.next=Newnode // This means the next pointer of the current node will start pointing to the new node.
Newnode.next.prev=Newnode // This means the previous of the node with element b start point to the newnode.
Hence every pointer is satisfied and the newnode with element x is inserted.

Question 17

The following function is executed on given list.
List has elements 4 → 5 → 3 → 2 → 1 → 2.

```
void print(struct node * head, int flag) {
   while (head != NULL) {
      if (flag) {
         head = head→ next;
         printf("%d", head→ data);
      } else {
         printf("%d", head→ data);
         head = head→ next;
      }
   }
   return;
}
```

Variable head stores the pointer to the head of the list.
What is the output of print (head, 1)?

(1) 5 3 1
(2) 5 3 2 1 2
(3) 5 2 2
(4) Null pointer dereference will occur

Answer

Option (4)

Question 18

Which of the following pseudo-code is correct when we are inserting the element x in the front of the single linked list and return the new head pointer.

```
I.   Create node;
     node → next = NULL;
     node → data = x;
     node → next = head;
     head=node;
     return head;
```

```
II.  Create node;
     node → data=x;
     node → next=head;
     head=node;
     return head;
```

```
III.Create node;
    node → x=data;
    node → next = NULL;
    return head;
```

(1) Only I, Ill

(2) Only II, III

(3) Only I, II

(4) All of I, II and III

Solution

Option (3) Statement I and II are correct.

The only difference in both the statements is in the first statement we have first assigned NULL to the next of the node and then changed it to the head. In the second statement, we have directly assigned next of node to the head without putting the NULL.

The third statement is incorrect node→ x=data is irrelevant because the element x we have to insert in the data field of the node.

Question 19

Which of the following pieces of code has the functionality of counting the number of nodes in the linked list?

```
(1)
struct node* temp = head;
int count=0;
while(temp != NULL){
temp = temp → next;
temp = head ;
count++; }
printf("%d", count);
```

```
(2)
struct node* temp = head;
int count=0;
while(temp != NULL){
temp = temp → next;
count++; }
printf("%d", count);
```

```
(3)
struct node* temp = head;
int count=0;
while(temp → next != NULL){
temp = temp → next;
count++; }
printf("%d", count);
```

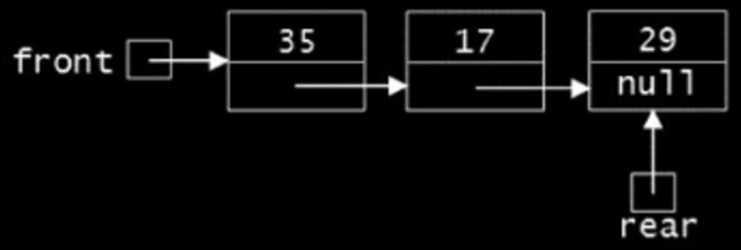
(4) None of these.

Solution

Option (2) For counting the number of nodes we have to move the temp pointer.
while(temp!=**NULL**) //The last node next stores the NULL
If we take the **while**(temp->next!=**NULL**) the last node will not be counted here.

Question 20

The diagram below suggests how we could implement a linked list in which we maintain a reference to both the first and last nodes in the linked list:



Which of the following operations would be inefficient to carry out when there are many elements in the linked list?
(1) Insertion at the end to which front refers
(2) Insertion at the end to which rear refers
(3) Deletion from the end to which front refers
(4) Deletion from the end to which rear refers

Solution

Sol: (4).

1. Insertion at the end where the front appears is possible because we have to point the new node → next to the node which is pointed by front initially and then make new node=front.
2. Insertion at the end where the rear appears is possible by putting the address of new node to the rear next.
3. Deletion from the end to which front refers is possible by putting the front to point front → next.
4. Deletion from the end to which rear refers is difficult because we have to move the front pointer to the second last node. In such a case if there are large numbers of elements then the deletion is not possible.

Question 21

```
Node* mystery( Node *head)
{
    if (head == NULL || head->next == NULL)
        return head;


    Node *node = head->next;

    head->next = node->next;
    free(node);

    mystery(head->next);

    return head;
}
```

What output can be expected from the function call `mystery(head)` called on the provided linked list?

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow \text{NULL}$

A. $1 \rightarrow 3 \rightarrow 5 \rightarrow 7 \rightarrow \text{NULL}$

B. $1 \rightarrow 3 \rightarrow 5 \rightarrow \text{NULL}$

C. $2 \rightarrow 4 \rightarrow 6 \rightarrow \text{NULL}$

D. None of the above

Answer A

Question 22

```
Node* fun(Node* head, int pos){

    if(pos<=0 || head == NULL) return head;

    if(pos == 1){

        Node *n = head->next;
        free(head);

        return n;
    }

    head->next  =  fun(head->next, pos-1);
    return head;
}
```

What output can be expected from the function call `fun(head,3)` called on the provided linked list?

4 → 3 → 7 → 9 → 2 → NULL

A. 4 → 7 → 9 → 2 → NULL

B. 4→ 3 → 9 → 2 → NULL

C. 4→ 3 → 7 → 2 → NULL

D. 4→ 3 → 7 → 2 → NULL

Answer B

Question 23

```
Node* removeNodes(Node* head) {

        if(head == NULL || head->next==NULL )
             return head;

        Node *n = removeNodes(head->next);

        if(n->data > head->data) return n;

        head->next = n;

        return head;

    }
```

What output can be expected from the function call `removeNodes(head)` called on the provided linked list?

5 → 2 → 13 → 3 → 8 → NULL

A. 5 → 2 → 13 → 8 → NULL

B. 13 → 3 → 8 → NULL

C. 13 → 8 → NULL

D. 13 → 3 → NULL

Answer C

Question 24

```
int recursive_alt_sum(Node* head)
{

    if (head == NULL ) return 0;
    else return head->data - recursive_alt_sum(head->next);

}
```

What output can be expected from the function call recursive_alt_sum(head) called on the provided linked list? $4 \rightarrow 2 \rightarrow 5 \rightarrow 2 \rightarrow \text{NULL}$

A. -5 **B. -1** **C. 5** **D. 9**

https://cse.buffalo.edu/~hungngo/classes/2014/Fall/250/assignments/mid2-sol.pdf

Answer

C

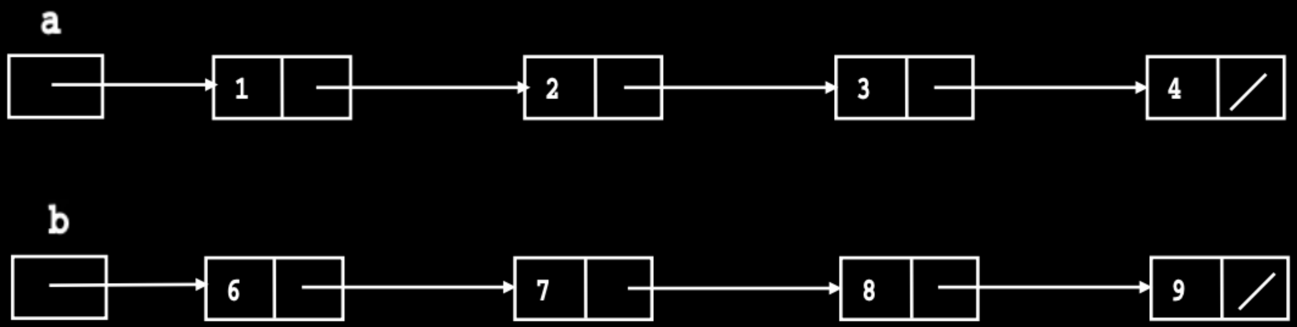
Question 25

```
Node* mystery (Node* a, Node* b)
{
    while (b){
        Node* t = a->next->next;
        a->next->next = b;
        a = b;
        b = t;
    }
    return a;

}
```

Consider the two linked list given below.

What will be the returned linked list ?



Answer: