# Linked List



**Question 1** CORRECT

What does the following function do for a given Linked List with first node as *head*?

```
void fun1(struct node* head)
{
  if(head == NULL)
    return;

  fun1(head->next);
  printf("%d  ", head->data);
}
```

A Prints all nodes of linked lists

Prints all nodes of linked list in reverse order

C Prints alternate nodes of Linked List

D Prints alternate nodes in reverse order

**Linked List**
**Discuss it**

**Question 1 Explanation:**
fun1() prints the given Linked List in reverse manner. For Linked List 1->2->3->4->5, fun1() prints 5->4->3->2->1. See http://www.geeksforgeeks.org/practice-questions-for-linked-list-and-recursion/

**Question 2** CORRECT

Which of the following points is/are true about Linked List data structure when it is compared with array

A Arrays have better cache locality that can make them better in terms of performance.

B It is easy to insert and delete elements in Linked List

C Random access is not allowed in a typical implementation of Linked Lists

D The size of array has to be pre-decided, linked lists can change their size any time.

All of the above

**Linked List**
**Discuss it**

**Question 2 Explanation:**
See http://www.geeksforgeeks.org/linked-list-vs-array/ for explanation.

**Question 3** CORRECT

Consider the following function that takes reference to head of a Doubly Linked List as parameter. Assume that a node of doubly linked list has previous pointer as *prev* and next pointer as *next*.

```
void fun(struct node **head_ref)
{
    struct node *temp = NULL;
    struct node *current = *head_ref;

    while (current !=  NULL)
    {
        temp = current->prev;
        current->prev = current->next;
        current->next = temp;
        current = current->prev;
    }

    if(temp != NULL )
        *head_ref = temp->prev;
}
```

Run on IDE

Assume that reference of head of following doubly linked list is passed to above function 1 <--> 2 <--> 3 <--> 4 <--> 5 <-->6. What should be the modified linked list after the function call?

A 2 <--> 1 <--> 4 <--> 3 <--> 6 <-->5

B 5 <--> 4 <--> 3 <--> 2 <--> 1 <-->6.

6 <--> 5 <--> 4 <--> 3 <--> 2 <--> 1.

D 6 <--> 5 <--> 4 <--> 3 <--> 1 <--> 2

**Linked List**
**Discuss it**

**Question 3 Explanation:**
The given function reverses the given doubly linked list. See Reverse a Doubly Linked List for details.

**Question 4** WRONG

Which of the following sorting algorithms can be used to sort a random linked list with minimum time complexity?

A Insertion Sort

Quick Sort

C Heap Sort

Merge Sort

**Linked List**
**Discuss it**

**Question 4 Explanation:**
Both Merge sort and Insertion sort can be used for linked lists. The slow random-access performance of a linked list makes other algorithms (such as quicksort) perform poorly, and others (such as heapsort) completely impossible. Since worst case time complexity of Merge Sort is O(nLogn) and Insertion sort is O(n^2), merge sort is preferred. See following for implementation of merge sort using Linked List. http://www.geeksforgeeks.org/merge-sort-for-linked-list/

**Question 5** CORRECT

The following function reverse() is supposed to reverse a singly linked list. There is one line missing at the end of the function.

```
/* Link list node */
struct node
{
    int data;
    struct node* next;
};

/* head_ref is a double pointer which points to head (or start) pointer
  of linked list */
static void reverse(struct node** head_ref)
{
    struct node* prev   = NULL;
    struct node* current = *head_ref;
    struct node* next;
    while (current != NULL)
    {
        next  = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    /*ADD A STATEMENT HERE*/
}
```

Run on IDE

What should be added in place of "/*ADD A STATEMENT HERE*/", so that the function correctly reverses a linked list.

A *head_ref = prev;

B *head_ref = current;

C *head_ref = next;

D *head_ref = NULL;

**Linked List**
**Discuss it**

**Question 5 Explanation:**
*head_ref = prev; At the end of while loop, the *prev* pointer points to the last node of original linked list. We need to change *head_ref so that the head pointer now starts pointing to the last node. See the following complete running program. 1

**Question 6** CORRECT

What is the output of following function for start pointing to first node of following linked list? 1->2->3->4->5->6

```
void fun(struct node* start)
{
  if(start == NULL)
    return;
  printf("%d  ", start->data);

  if(start->next != NULL )
    fun(start->next->next);
  printf("%d  ", start->data);
}
```

Run on IDE

A 1 4 6 6 4 1

B 1 3 5 1 3 5

C 1 2 3 5

1 3 5 5 3 1

**Linked List**
**Discuss it**

**Question 6 Explanation:**
fun() prints alternate nodes of the given Linked List, first from head to end, and then from end to head. If Linked List has even number of nodes, then skips the last node.

**Question 7** CORRECT

The following C function takes a simply-linked list as input argument. It modifies the list by moving the last element to the front of the list and returns the modified list. Some part of the code is left blank. Choose the correct alternative to replace the blank line.

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

Node *move_to_front(Node *head)
{
  Node *p, *q;
  if ((head == NULL: || (head->next == NULL))
```

```
    return head;
  q = NULL; p = head;
  while (p-> next !=NULL)
  {
    q = p;
    p = p->next;
  }
  ____________________________
  return head;
}
```

Run on IDE

A q = NULL; p->next = head; head = p;

B q->next = NULL; head = p; p->next = head;

C head = p; p->next = q; q->next = NULL;

q->next = NULL; p->next = head; head = p;

**Linked List**
**Discuss it**

**Question 7 Explanation:**
See question 1 of http://www.geeksforgeeks.org/data-structures-and-algorithms-set-24/

**Question 8** CORRECT

The following C function takes a single-linked list of integers as a parameter and rearranges the elements of the list. The function is called with the list containing the integers 1, 2, 3, 4, 5, 6, 7 in the given order. What will be the contents of the list after the function completes execution?

```
struct node
{
  int value;
  struct node *next;
};
void rearrange(struct node *list)
{
  struct node *p, * q;
  int temp;
  if ((!list) || !list->next)
      return;
  p = list;
  q = list->next;
  while(q)
  {
     temp = p->value;
     p->value = q->value;
```

```
    q->value = temp;
    p = q->next;
    q = p?p->next:0;
  }
}
```

Run on IDE

A 1,2,3,4,5,6,7

2,1,4,3,6,5,7

C 1,3,2,5,4,7,6

D 2,3,4,5,6,7,1

**Linked List**
**Discuss it**

**Question 8 Explanation:**
The function rearrange() exchanges data of every node with its next node. It starts exchanging data from the first node itself.

**Question 9** CORRECT

In the worst case, the number of comparisons needed to search a singly linked list of length n for a given element is (GATE CS 2002)

A log 2 n

B n/2

C log 2 n – 1

n

**Linked List**
**Discuss it**

**Question 9 Explanation:**
In the worst case, the element to be searched has to be compared with all elements of linked list.

**Question 10** WRONG

Suppose each set is represented as a linked list with elements in arbitrary order. Which of the operations among union, intersection, membership, cardinality will be the slowest? (GATE CS 2004)

A union only

B intersection, membership

membership, cardinality

union, intersection

**Linked List**
**Discuss it**

**Question 10 Explanation:**
For getting intersection of L1 and L2, search for each element of L1 in L2 and print the elements we find in L2. There can be many ways for getting union of L1 and L2. One of them is as follows a) Print all the nodes of L1 and print only those which are not present in L2. b) Print nodes of L2. All of these methods will require more operations than intersection as we have to process intersection node plus other nodes.

**You have completed** 10/31 questions .
Your accuracy is 80%.
1 2 3 4



**Company Wise Coding Practice** **Topic Wise Coding Practice**