**1.write a program to delete the middle element of linked list**

Code:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

void deleteMiddle(struct Node\*\* head) {

if (\*head == NULL || (\*head)->next == NULL) {

free(\*head);

\*head = NULL;

return;

}

struct Node \*slow = \*head, \*fast = \*head, \*prev = NULL;

while (fast != NULL && fast->next != NULL) {

fast = fast->next->next;

prev = slow;

slow = slow->next;

}

prev->next = slow->next;

free(slow);

}

void push(struct Node\*\* head, int new\_data) {

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

new\_node->data = new\_data;

new\_node->next = \*head;

\*head = new\_node;

}

void printList(struct Node\* head) {

while (head != NULL) {

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

head = head->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

push(&head, 5);

push(&head, 4);

push(&head, 3);

push(&head, 2);

push(&head, 1);

printf("Original list: ");

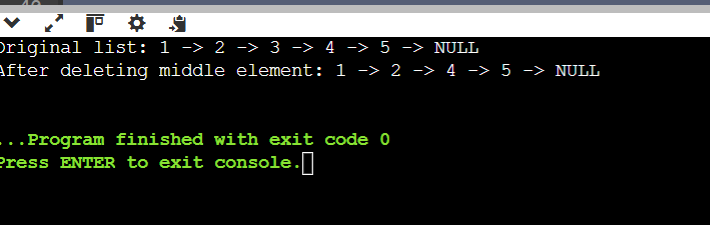
printList(head);

deleteMiddle(&head);

printf("After deleting middle element: ");

printList(head);

return 0;

}

**2. Write a program to merge a linked list with another linked list at alternate positions.**

CODE:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

void mergeAlternate(struct Node\* list1, struct Node\* list2) {

struct Node \*curr1 = list1, \*curr2 = list2, \*next1, \*next2;

while (curr1 != NULL && curr2 != NULL) {

next1 = curr1->next;

next2 = curr2->next;

curr1->next = curr2;

curr2->next = next1;

curr1 = next1;

curr2 = next2;

}

}

void push(struct Node\*\* head, int new\_data) {

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

new\_node->data = new\_data;

new\_node->next = \*head;

\*head = new\_node;

}

void printList(struct Node\* head) {

while (head != NULL) {

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

head = head->next;

}

printf("NULL\n");

}

int main() {

struct Node \*list1 = NULL, \*list2 = NULL;

push(&list1, 7);

push(&list1, 5);

push(&list1, 3);

push(&list1, 1);

push(&list2, 8);

push(&list2, 6);

push(&list2, 4);

push(&list2, 2);

printf("List 1: ");

printList(list1);

printf("List 2: ");

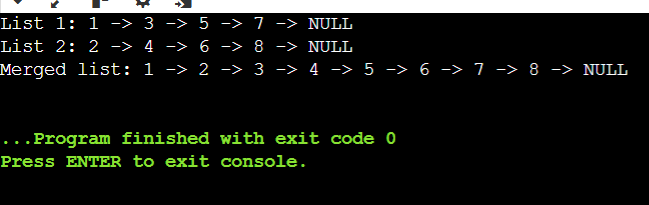
printList(list2);

mergeAlternate(list1, list2);

printf("Merged list: ");

printList(list1);

return 0;

}

**3.write a program to swap k th node from the beginning with the k th node from the end of the list.**

Code:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

void push(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;

}

int countNodes(struct Node\* head) {

int count = 0;

while (head != NULL) {

head = head->next;

count++;

}

return count;

}

void swapKthNode(struct Node\*\* head, int k) {

int n = countNodes(\*head);

if (k > n || 2 \* k - 1 == n) return;

struct Node \*x = \*head, \*x\_prev = NULL;

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

x\_prev = x;

x = x->next;

}

struct Node \*y = \*head, \*y\_prev = NULL;

for (int i = 1; i < n - k + 1; i++) {

y\_prev = y;

y = y->next;

}

if (x\_prev) x\_prev->next = y;

if (y\_prev) y\_prev->next = x;

struct Node\* temp = x->next;

x->next = y->next;

y->next = temp;

if (k == 1) \*head = y;

if (k == n) \*head = x;

}

void printList(struct Node\* head) {

while (head != NULL) {

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

head = head->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

for (int i = 8; i >= 1; i--) push(&head, i);

printf("Original list: ");

printList(head);

int k = 3;

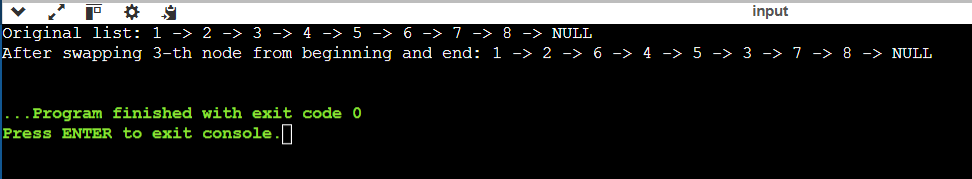
swapKthNode(&head, k);

printf("After swapping %d-th node from beginning and end: ", k);

printList(head);

return 0;

}



**3.write a program to convert a 2d matrix into a linked list matrix such that each node is linked towards next right and down node and display it.**

Code:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* right;

struct Node\* down;

};

struct Node\* createNode(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->right = NULL;

newNode->down = NULL;

return newNode;

}

struct Node\* constructLinkedMatrix(int mat[][3], int rows, int cols) {

struct Node\* head = NULL;

struct Node\* rowHead = NULL;

struct Node\* prevRowHead = NULL;

struct Node\* temp = NULL;

for (int i = 0; i < rows; i++) {

rowHead = NULL;

temp = NULL;

for (int j = 0; j < cols; j++) {

struct Node\* newNode = createNode(mat[i][j]);

if (head == NULL) head = newNode;

if (rowHead == NULL) rowHead = newNode;

else temp->right = newNode;

if (prevRowHead != NULL) {

prevRowHead->down = newNode;

prevRowHead = prevRowHead->right;

}

temp = newNode;

}

prevRowHead = rowHead;

}

return head;

}

void displayLinkedMatrix(struct Node\* head) {

struct Node\* rowHead = head;

while (rowHead != NULL) {

struct Node\* temp = rowHead;

while (temp != NULL) {

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

temp = temp->right;

}

printf("\n");

rowHead = rowHead->down;

}

}

int main() {

int mat[3][3] = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

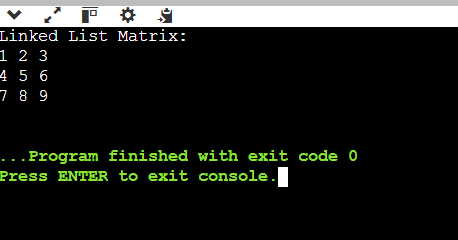
struct Node\* head = constructLinkedMatrix(mat, 3, 3);

printf("Linked List Matrix:\n");

displayLinkedMatrix(head);

return 0;

}



**4.write a program to remove all the odd positioned elements from the circular linked list.**

Code:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

void push(struct Node\*\* head, int data) {

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

struct Node\* temp = \*head;

new\_node->data = data;

new\_node->next = \*head;

if (\*head != NULL) {

while (temp->next != \*head)

temp = temp->next;

temp->next = new\_node;

} else {

new\_node->next = new\_node;

}

\*head = new\_node;

}

void removeOddPositioned(struct Node\*\* head) {

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

struct Node\* prev = \*head;

struct Node\* curr = (\*head)->next;

int position = 2;

while (curr != \*head) {

if (position % 2 != 0) {

prev->next = curr->next;

free(curr);

curr = prev->next;

} else {

prev = curr;

curr = curr->next;

}

position++;

}

if (position % 2 != 0) {

prev->next = (\*head)->next;

free(\*head);

\*head = prev->next;

}

}

void printList(struct Node\* head) {

if (head == NULL) return;

struct Node\* temp = head;

do {

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

temp = temp->next;

} while (temp != head);

printf("(back to head)\n");

}

int main() {

struct Node\* head = NULL;

for (int i = 6; i >= 1; i--)

push(&head, i);

printf("Original Circular Linked List:\n");

printList(head);

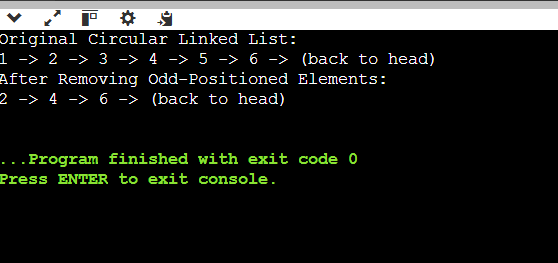
removeOddPositioned(&head);

printf("After Removing Odd-Positioned Elements:\n");

printList(head);

return 0;

}



**5.write a program to merge two linked list and sort them.**

Code:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

void push(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;

}

struct Node\* mergeSortedLists(struct Node\* head1, struct Node\* head2) {

struct Node dummy;

struct Node\* tail = &dummy;

dummy.next = NULL;

while (head1 != NULL && head2 != NULL) {

if (head1->data <= head2->data) {

tail->next = head1;

head1 = head1->next;

} else {

tail->next = head2;

head2 = head2->next;

}

tail = tail->next;

}

if (head1 != NULL) tail->next = head1;

if (head2 != NULL) tail->next = head2;

return dummy.next;

}

void sortList(struct Node\*\* head) {

struct Node\* sorted = NULL;

struct Node\* current = \*head;

while (current != NULL) {

struct Node\* next = current->next;

struct Node\*\* insert\_pos = &sorted;

while (\*insert\_pos != NULL && (\*insert\_pos)->data < current->data) {

insert\_pos = &(\*insert\_pos)->next;

}

current->next = \*insert\_pos;

\*insert\_pos = current;

current = next;

}

\*head = sorted;

}

void mergeAndSort(struct Node\*\* head1, struct Node\*\* head2) {

struct Node\* merged = mergeSortedLists(\*head1, \*head2);

sortList(&merged);

\*head1 = merged;

\*head2 = NULL;

}

void printList(struct Node\* head) {

while (head != NULL) {

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

head = head->next;

}

printf("NULL\n");

}

int main() {

struct Node\* list1 = NULL;

struct Node\* list2 = NULL;

push(&list1, 5);

push(&list1, 3);

push(&list1, 1);

push(&list2, 6);

push(&list2, 4);

push(&list2, 2);

printf("List 1: ");

printList(list1);

printf("List 2: ");

printList(list2);

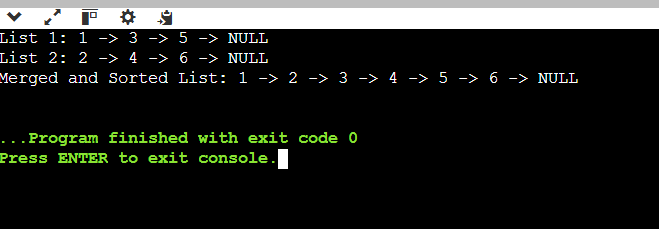
mergeAndSort(&list1, &list2);

printf("Merged and Sorted List: ");

printList(list1);

return 0;

}



6.write a program to reverse a circular linked list.

Code:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

void push(struct Node\*\* head, int data) {

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

struct Node\* temp = \*head;

new\_node->data = data;

new\_node->next = \*head;

if (\*head != NULL) {

while (temp->next != \*head)

temp = temp->next;

temp->next = new\_node;

} else {

new\_node->next = new\_node;

}

\*head = new\_node;

}

void reverse(struct Node\*\* head) {

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

struct Node \*prev = NULL, \*current = \*head, \*next = NULL;

do {

next = current->next;

current->next = prev;

prev = current;

current = next;

} while (current != \*head);

(\*head)->next = prev;

\*head = prev;

}

void printList(struct Node\* head) {

if (head == NULL) return;

struct Node\* temp = head;

do {

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

temp = temp->next;

} while (temp != head);

printf("(back to head)\n");

}

int main() {

struct Node\* head = NULL;

push(&head, 5);

push(&head, 4);

push(&head, 3);

push(&head, 2);

push(&head, 1);

printf("Original Circular Linked List:\n");

printList(head);

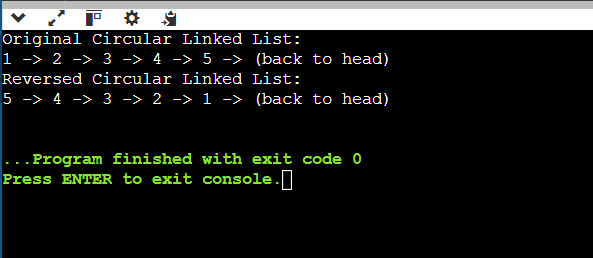
reverse(&head);

printf("Reversed Circular Linked List:\n");

printList(head);

return 0;

}



**7. write a program to modify a linked list such that all the even numbers appears before the odd numbers in the list.**

Code:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

void push(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 rearrangeEvenOdd(struct Node\*\* head) {

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

struct Node\* evenHead = NULL;

struct Node\* oddHead = NULL;

struct Node\* evenTail = NULL;

struct Node\* oddTail = NULL;

struct Node\* curr = \*head;

while (curr != NULL) {

if (curr->data % 2 == 0) {

if (evenHead == NULL) {

evenHead = evenTail = curr;

} else {

evenTail->next = curr;

evenTail = evenTail->next;

}

} else {

if (oddHead == NULL) {

oddHead = oddTail = curr;

} else {

oddTail->next = curr;

oddTail = oddTail->next;

}

}

curr = curr->next;

}

if (evenTail != NULL) {

evenTail->next = oddHead;

}

if (oddTail != NULL) {

oddTail->next = NULL;

}

\*head = (evenHead != NULL) ? evenHead : oddHead;

}

void printList(struct Node\* head) {

while (head != NULL) {

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

head = head->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

push(&head, 11);

push(&head, 10);

push(&head, 9);

push(&head, 6);

push(&head, 4);

push(&head, 1);

printf("Original List:\n");

printList(head);

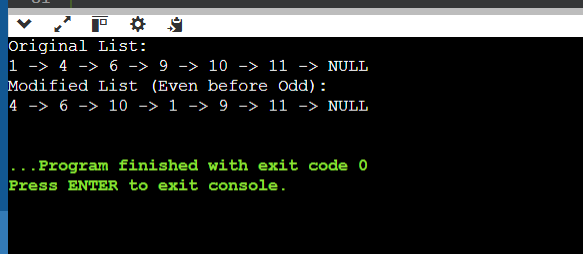
rearrangeEvenOdd(&head);

printf("Modified List (Even before Odd):\n");

printList(head);

return 0;

}



**8.write a program such that n nodes are deleted after m th index in the linked list**.

Code:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

void push(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 deleteAfterM(struct Node\*\* head, int m, int n) {

struct Node\* current = \*head;

for (int i = 1; i < m && current != NULL; i++) {

current = current->next;

}

if (current == NULL) return;

struct Node\* temp;

for (int i = 0; i < n && current->next != NULL; i++) {

temp = current->next;

current->next = temp->next;

free(temp);

}

}

void printList(struct Node\* head) {

while (head != NULL) {

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

head = head->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

for (int i = 8; i >= 1; i--) {

push(&head, i);

}

printf("Original List:\n");

printList(head);

int m = 3;

int n = 2;

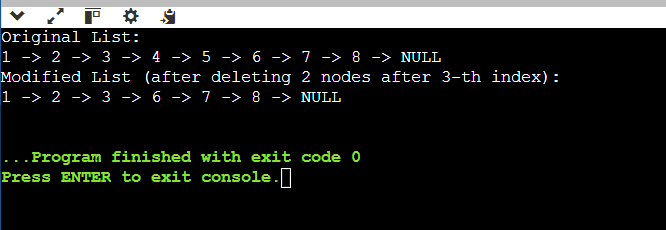
deleteAfterM(&head, m, n);

printf("Modified List (after deleting %d nodes after %d-th index):\n", n, m);

printList(head);

return 0;

}



9.write a program to split a circular linked list into two simple linked list.

Code:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

void push(struct Node\*\* head, int data) {

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

new\_node->data = data;

if (\*head == NULL) {

new\_node->next = new\_node;

\*head = new\_node;

} else {

struct Node\* temp = \*head;

while (temp->next != \*head) {

temp = temp->next;

}

temp->next = new\_node;

new\_node->next = \*head;

}

}

void splitCircularList(struct Node\* head, struct Node\*\* head1, struct Node\*\* head2) {

if (head == NULL || head->next == head) {

\*head1 = head;

\*head2 = NULL;

return;

}

struct Node\* slow = head;

struct Node\* fast = head;

while (fast->next != head && fast->next->next != head) {

slow = slow->next;

fast = fast->next->next;

}

\*head1 = head;

\*head2 = slow->next;

slow->next = head; // Breaking the first list

fast->next = \*head2; // Completing the second list circularly

}

void printList(struct Node\* head) {

if (head == NULL) return;

struct Node\* temp = head;

do {

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

temp = temp->next;

} while (temp != head);

printf("(back to head)\n");

}

int main() {

struct Node\* head = NULL;

struct Node\* head1 = NULL;

struct Node\* head2 = NULL;

for (int i = 8; i >= 1; i--) {

push(&head, i);

}

printf("Original Circular Linked List:\n");

printList(head);

splitCircularList(head, &head1, &head2);

printf("First Split Circular Linked List:\n");

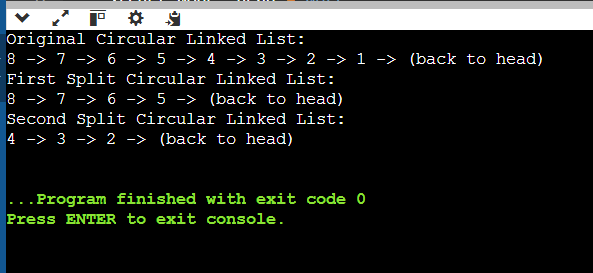
printList(head1);

printf("Second Split Circular Linked List:\n");

printList(head2);

return 0;

}



**10.write a c program to print the occurrence of each number in the linked list.**

Code:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

void push(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 printOccurrences(struct Node\* head) {

struct Node\* current = head;

struct Node\* temp;

int count;

while (current != NULL) {

count = 1;

temp = current->next;

while (temp != NULL) {

if (current->data == temp->data) {

count++;

temp->data = -1; // Mark as counted

}

temp = temp->next;

}

if (current->data != -1) {

printf("Number %d occurs %d time(s)\n", current->data, count);

}

current = current->next;

}

}

void printList(struct Node\* head) {

while (head != NULL) {

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

head = head->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

push(&head, 3);

push(&head, 2);

push(&head, 1);

push(&head, 2);

push(&head, 3);

push(&head, 4);

printf("Linked List:\n");

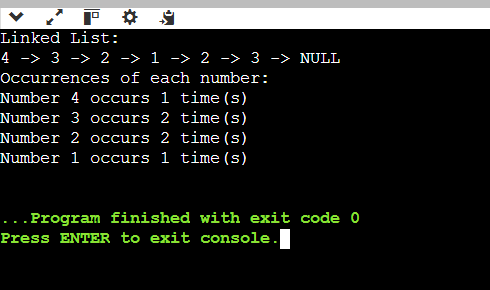
printList(head);

printf("Occurrences of each number:\n");

printOccurrences(head);

return 0;

}



**11.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);  
}

1.Prints all nodes of linked lists

2.Prints all nodes of linked list in reverse order

3.Prints alternate nodes of Linked List

4.Prints alternate nodes in reverse order

**Explanation :**

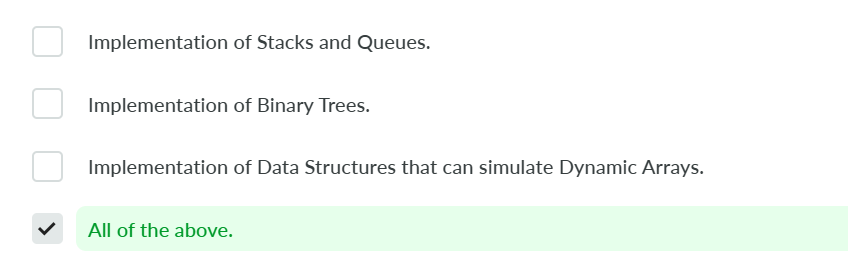
This function **fun1** recursively prints the elements of a linked list in reverse order.

It first checks if the head is NULL, and if it is, it returns without doing anything. Otherwise, it calls the **fun1** function recursively with the next node in the linked list (**head->next**). This continues until the last node is reached, which is the node whose **next** pointer is **NULL**.

At this point, the function starts returning, and as it returns it prints the value of each node in the linked list starting from the last node, working its way backwards to the first node. This is achieved through the **printf** statement which prints the value of **head->data**.

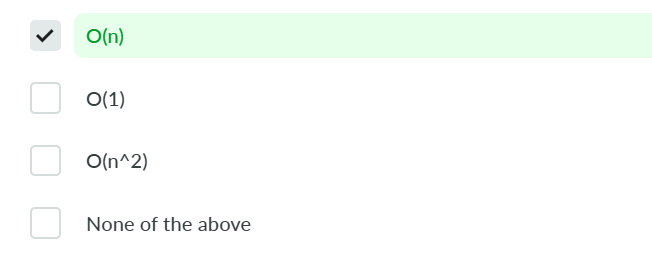
Thus, when this function is called with the head of a linked list as an argument, it will print the values of the nodes in reverse order

**12. Which of the following can be done with LinkedList?**



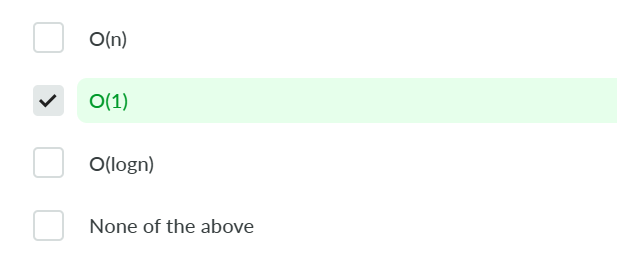
**Answer: D)** Explanation: All of the above operations can be done with a linked list.

**13. What is the time complexity of a program to reverse a linked list?**



**Answer: A)** Explanation: We can reverse a linked list in O(n) time complexity.

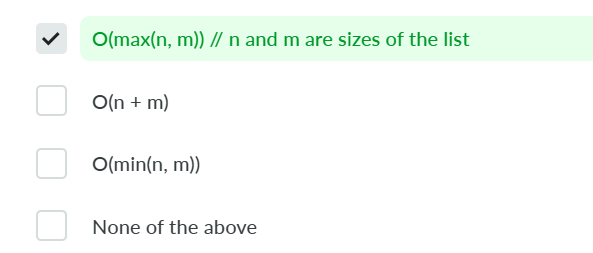
**14.What is the time complexity to insert an element to the front of a LinkedList(head pointer given)?**



**Correct Answer**

**Answer: B)** Explanation: We set the next node to the head of the list, and then return that node as the new head.

**15.What is the time complexity of adding 2 numbers as a linked list?**

****

**Correct Answer**

**Answer: A)** Explanation: We can add 2 linked lists in O(max(n, m)) where n and m are the lengths of the 2 linked lists respectively.

**16. What is the output of following function in which start is pointing to the first node of the following linked list 1->2->3->4->5->6 ?**

void fun(**struct** **node**\* start)

{

**if**(start == NULL)

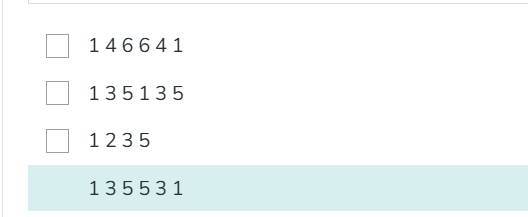
**return**;

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

**if**(start->next != NULL )

fun(start->next->next);

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

}

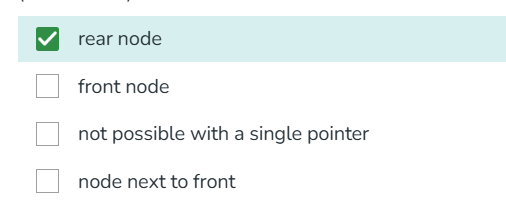
**Explanation**

The function prints the data of the current node and then recursively calls itself with the second next node (i.e., start->next->next).

So, it prints the data of every alternate node of the linked list i.e 1 3 5, and then, since the next->next of 5 is null, it returns and prints the data of the current node, so it then prints 5 3 1.

Therefore, for the given linked list 1->2->3->4->5->6, the function would print 1 3 5 5 3 1.

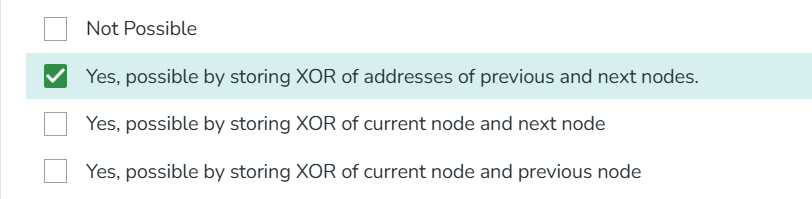
**17. A circularly linked list is used to represent a Queue. A single variable p is used to access the Queue. To which node should p point such that both the operations enQueue and deQueue can be performed in constant time?**

****

**Explanation**

Answer is not "(b) front node", as we can not get rear from front in O(1), but if p is rear we can implement both enQueue and deQueue in O(1) because from rear we can get front in O(1). Below are sample functions. Note that these functions are just sample are not working. Code to handle base cases is missing.

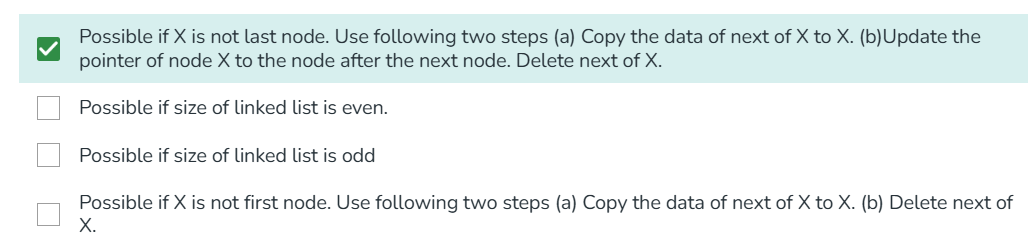
**18.** **Is it possible to create a doubly linked list using only one pointer with every node.**

****

**Explanation**

Yes, it is possible to implement a doubly linked list using a single pointer per node, by storing the XOR of the addresses of the previous and next nodes.

**19. Given pointer to a node X in a singly linked list. Only one pointer is given, pointer to head node is not given, can we delete the node X from given linked list?**

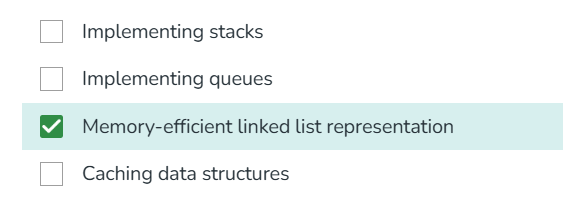
****

**Explanation**

Following are simple steps.

struct node \*temp = X->next;  
 X->data = temp->data;  
 X->next = temp->next;  
 free(temp);

**20. Which of the following is an application of XOR-linked lists?**

****

**Explanation**

XOR linked lists are a memory-efficient representation of linked lists. They store the XOR combination of the addresses of the previous and next nodes, reducing the memory overhead compared to traditional linked lists.