



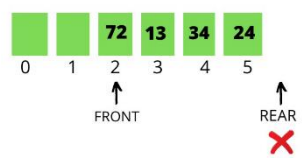
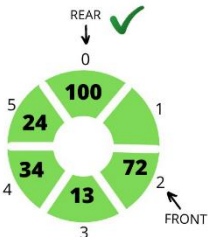
6	<p>For the given sparse matrix, write the C-representation of the header and element nodes. Represent the sparse matrix using linked list.</p> $\begin{pmatrix} 2 & 0 & 0 & 0 \\ 4 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 1 \\ 0 & 0 & 6 & 0 \end{pmatrix}$ <p><b>The reverse() function is supposed to reverse a singly linked list. One line is missing at the end of the function.</b></p> <pre> struct node { int data; struct node *next; };  /* head_refer is a double pointer that points to the head (or start) pointer of linked list */  void reverse( struct node** head_refer ) {     struct node* prev = NULL;     struct node* current = *head_refer;     struct node* next;     while (current != NULL)     {         next = current-&gt;next;         current-&gt;next = prev;         prev = current;         current = next;     }     /*ADD A STATEMENT HERE*/ } </pre>	[10]	CO3	L3
---	---	------	-----	----

**Internal Assessment Test 2 – Dec 2022 (Solution)**

Sub:	Data Structures and Applications	Sub Code:	21CS32	Branch:	CSE
<b>Solution</b>					

Answer any FIVE FULL Questions

**MARKS**    **CO**    **RB**  
**S**            **T**

1	<p><b>Question:</b>  List the advantages of circular queue over ordinary queue? With suitable C-functions simulate the working of circular Queue of integers using Arrays.  Suppose a queue is maintained by a circular array queue with N=12 memory cells.  Find the number of elements in the queue if</p> <p>i)     FRONT =4 REAR =8 <b>Answer : 4</b>  ii)     FRONT =10 REAR = 3 <b>Answer : 5</b>  iii)     FRONT =5 REAR =6 and then two elements are deleted.</p> <p><b>Answer:</b></p> <p>Advantages of circular queue</p> <div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="text-align: center;">  <p>Insertion not possible !</p> </div> <div style="text-align: center;">  <p>Insertion possible !</p> </div> </div> <ul style="list-style-type: none"> <li><b>Easier for insertion-deletion:</b> In the circular queue, elements can be inserted easily if there are vacant locations until it is not fully occupied, whereas in the case of a linear queue insertion is not possible once the rear reaches the last index even if there are empty locations present in the queue</li> <li><b>Efficient utilization of memory:</b> In the circular queue, there is no wastage of memory as it uses the unoccupied space, and memory is used properly in a valuable and effective manner as compared to a linear queue.</li> <li><b>Ease of performing operations:</b> In the linear queue, FIFO is followed, so the element inserted first is the element to be deleted first. This is not the scenario in the case of the circular queue as the rear and front are not fixed so the order of insertion-deletion can be changed, which is very useful.</li> </ul> <p>With suitable C-functions simulate the working of circular Queue of integers using Arrays.</p> <pre> #define capacity 6  int queue[capacity]; int front = -1, rear = -1;  // Here we check if the Circular queue is full or not int checkFull () {     if ((front == rear + 1)    (front == 0 &amp;&amp; rear == capacity - 1))     { </pre>	[10]	CO2	L3
---	---	------	-----	----

```

    return 1;
}
return 0;
}

// Here we check if the Circular queue is empty or not
int checkEmpty ()
{
    if (front == -1)
    {
        return 1;
    }
    return 0;
}

// Addition in the Circular Queue
void enqueue (int value)
{
    if (checkFull ())
        printf ("Overflow condition\n");

    else
    {
        if (front == -1)
            front = 0;

        rear = (rear + 1) % capacity;
        queue[rear] = value;
        printf ("%d was enqueued to circular queue\n", value);
    }
}

// Removal from the Circular Queue
int dequeue ()
{
    int variable;
    if (checkEmpty ())
    {
        printf ("Underflow condition\n");
        return -1;
    }
    else
    {
        variable = queue[front];
        if (front == rear)
        {
            front = rear = -1;
        }
        else
        {
            front = (front + 1) % capacity;
        }
        printf ("%d was dequeued from circular queue\n", variable);
        return 1;
    }
}
}

```

```
// Display the queue
void print ()
{
    int i;
    if (checkEmpty ())
        printf ("Nothing to dequeue\n");
    else
    {
        printf ("\nThe queue looks like: \n");
        for (i = front; i != rear; i = (i + 1) % capacity)
            {
                printf ("%d ", queue[i]);
            }
        printf ("%d \n\n", queue[i]);
    }
}
```

- i) FRONT =4 REAR =8 **Answer : 4**
- ii) FRONT =10 REAR = 3 **Answer : 5**
- iii) FRONT =5 REAR =6 and then two elements are deleted.

**Q-1**

0	1	2	3	4	5	6	7	8	9	10	11
				F				R			

**Q-2**

0	1	2	3	4	5	6	7	8	9	10	11
			R							F	

**Q-3**

0	1	2	3	4	5	6	7	8	9	10	11
					F	R					

- 2 Write C functions to perform the following operations in a SLL:
- i) Assume a four node single linked list with data values 15,25,40,50
  - ii) Insert a node with data value '60' at the end of the list.
  - iii) Insert a node with data value 30 in between the nodes 25 and 40
  - iv) Delete a node with data value '40'
  - v) Search node with data value '25'

**Solution:**

- i) Assume a four node single linked list with data values 15,25,40,50
- ii) Insert a node with data value '60' at the end of the list.

[10]

CO1 L3

```

struct node *insert_end(struct node *start)
{
    struct node *ptr, *new_node;
    int num;
    printf("\n Enter the data : ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node -> data = num;
    new_node -> next = NULL;
    ptr = start;
    while(ptr -> next != NULL)
    ptr = ptr -> next;
    ptr -> next = new_node;
    return start;
}

```

iii) Insert a node with data value 30 in between the nodes 25 and 40

```

struct node *insert_atter(struct node *start)
{
    struct node *new_node, *ptr, *preptr;
    int num, val;
    printf("\n Enter the data : ");
    scanf("%d", &num);
    printf("\n Enter the value after which the data has to be inserted ");
    scanf("%d", &val);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node -> data = num;
    ptr = start;
    preptr = ptr;
    while(preptr -> data != val)
    {
        preptr = ptr;
        ptr = ptr -> next;
    }
    preptr -> next=new_node;
    new_node -> next = ptr;
    return start;
}

```

iv) Delete a node with data value '40'

```

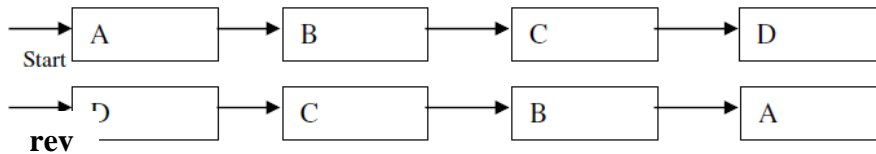
struct node *delete_node(struct node *start)
{
    struct node *ptr, *preptr;
    int val;
    printf("\n Enter the value of the node which has to be deleted ");
    scanf("%d", &val);
    ptr = start;
    if(ptr -> data == val)
    {
        start = delete_beg(start);
        return start;
    }
    else
    {
        while(ptr -> data != val)
        {
            preptr = ptr;
            ptr = ptr -> next;
        }
        preptr -> next = ptr -> next;
        free(ptr);
        return start;
    }
}

```

v) Search node with data value '25'

3 Write C functions to perform the following operations in the SLL in figure below:

- i. To count number of nodes in the given singly linked list.
- ii. To reverse direction of singly linked list (as shown below).
- iii. To concatenate the two singly linked list.



Count:

```
void print(){
    struct node* temp = head;
    int count=0;
    /* Traverse the linked list and maintain the count. */
    while(temp != NULL){
        temp = temp->next;
        count++;
    }
    printf("\n Total no. of nodes is %d",count);
}
```

Reverse a List

```
void reverse()
{
    // Initialize current, previous and next pointers
    Node* current = head;
    Node *prev = NULL, *next = NULL;
    while (current != NULL) {
        // Store next
        next = current->next;
        // Reverse current node's pointer
        current->next = prev;
        // Move pointers one position ahead.
        prev = current;
        current = next;
    }
    head = prev;
}
```

Concatenation:

```
void Concat(struct Node *first, struct Node *second)
{
    struct Node *p = first;
    while (p->next != NULL)
    {
```

[10] CO3 L3

	<pre> p = p-&gt;next; } p-&gt;next = second; second = NULL; } </pre>			
4	<p>Describe the doubly linked list with advantages and disadvantages. Write necessary C- functions to perform the following:</p> <ol style="list-style-type: none"> <li>i. Insert a node at the front of DLL</li> <li>ii. Delete a node from the front of DLL</li> <li>iii. Insert a node from a DLL before a node with a given value.</li> <li>iv. Delete a node from a DLL before a node with a given value.</li> </ol> <ol style="list-style-type: none"> <li>i. Insert a node at the front of DLL <pre> struct node *insert_beg(struct node *start) {     struct node *new_node;     int num;     printf("\n Enter the data : ");     scanf("%d", &amp;num);     new_node = (struct node *)malloc(sizeof(struct node));     new_node-&gt;data = num;      start-&gt;prev = new_node;     new_node-&gt;next = start;     new_node-&gt;prev = NULL;     start = new_node;     return start; } </pre> </li> <li>ii. Delete a node from the front of DLL <pre> struct node *delete_beg(struct node *start) {     struct node *ptr;     ptr = start;     start = start-&gt;next;     start-&gt;prev = NULL;     free(ptr);     return start; } </pre> </li> <li>iii. Insert a node from a DLL before a node with a given value. <pre> struct node *insert_before(struct node *start) {     struct node *new_node, *ptr;     int num, val;     printf("\n Enter the data : ");     scanf("%d", &amp;num);     printf("\n Enter the value before which the data has to be inserted:");     scanf("%d", &amp;val);     new_node = (struct node *)malloc(sizeof(struct node));     new_node-&gt;data = num;     ptr = start;     while(ptr-&gt;data != val)         ptr = ptr-&gt;next;     new_node-&gt;next = ptr;     new_node-&gt;prev = ptr-&gt;prev;     ptr-&gt;prev-&gt;next = new_node;     ptr-&gt;prev = new_node;     return start; } </pre> </li> <li>iv. Delete a node from a DLL before a node with a given value.</li> </ol>	[10]	CO3	L2



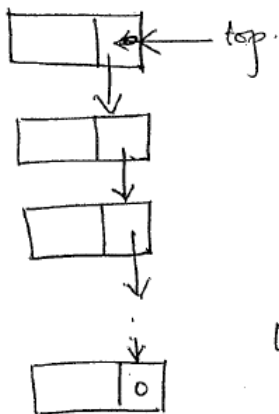
```

struct node *delete_before(struct node *start)
{
    struct node *ptr, *temp;
    int val;
    printf("\n Enter the value before which the node has to delete");
    scanf("%d", &val);
    ptr = start;
    while(ptr->data != val)
        ptr = ptr->next;
    temp = ptr->prev;
    if(temp == start)
        start = delete_beg(start);
    else
    {
        ptr->prev = temp->prev;
        temp->prev->next = ptr;
    }
    free(temp);
    return start;
}

```

5 Demonstrate the various operations performed in a Linked Stack with suitable C-function.

If a stacks and queues uses linked list for representation, you called them linked stack and queues.



Linked stack.

Initial condition for stack:

$$top[i] = NULL ; 0 \leq i < MAX\_STACKS$$

Boundary condition is:

$$top[i] = NULL \text{ iff. stack is empty.}$$

Operations on stack:

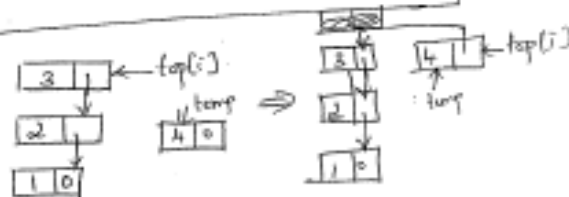
[10] CO3 L2

### Push:

- To insert an item to a stack.
- \* Create a new node, temp using malloc function.
- \* Place ~~data~~ <sup>item</sup> in the data field and top in the link field.
- \* top is then made to point to temp.

```
void push(int i, element item)
{
    stackptr temp;
    temp = (stackptr) malloc(sizeof(stackptr));
    temp->data = item;
    temp->link = top[i];
    top[i] = temp;
}
```

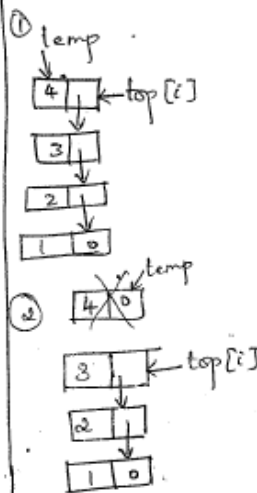
Figure:



### Pop:

- \* Pop returns the top element and changes top to point the address contained in its link field.
- \* The removed node is then freed and item is returned.

```
element pop(int i)
{
    stackptr temp = top[i];
    element item;
    if(!temp)
        return stackEmpty();
    item = temp->data;
    top[i] = temp->link;
    free(temp);
    return item;
}
```



(25)

6 For the given sparse matrix, write the C-representation of the header and element nodes. Represent the sparse matrix using linked list.

$$\begin{pmatrix} 2 & 0 & 0 & 0 \\ 4 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 1 \\ 0 & 0 & 6 & 0 \end{pmatrix}$$

**The reverse() function is supposed to reverse a singly linked list. One line is missing at the end of the function.**

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

/* head_refer is a double pointer that points to the head (or
start) pointer of linked list */

void reverse( struct node** head_refer )
{
    struct node* prev = NULL;
    struct node* current = *head_refer;
    struct node* next;
    while (current != NULL)
    {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    /*ADD A STATEMENT HERE*/
}
```

**Answer: head\_ref=prev;**