1.Write a program to reverse double liked list

This is really easy, just keep swapping the prev and next pointers and at the end swap the head and the tail:)

```
void reverse()
{
        mynode *cur, *temp, *save_next;
        if(head==tail)return;
        if(head==NULL || tail==NULL)return;
        for(cur=head;cur!=NULL;)
        {
                printf("\ncur->value : [%d]\n",cur->value);
                temp=cur->next;
                save_next=cur->next;
                cur->next=cur->prev;
                cur->prev=temp;
                cur=save_next;
        }
        temp=head;
        head=tail;
        tail=temp;
}
```

2.Given only a pointer to a node to be deleted in a singly linked list, how do you delete it?

This is a very good interview question

The solution to this is to copy the data from the next node into this node and delete the next node!. Ofcourse this wont work if the node to be deleted is the last node. Mark it as dummy in that case.

3.How would you detect a loop in a linked list? Write a C program to detect a loop in a linked list.

Have 2 pointers to start of the linked list. Increment one pointer by 1 node and the other by 2 nodes. If there's a loop, the 2nd pointer will
meet the 1st pointer somewhere. If it does, then you know there's one.
Here is some code

```
p=head;
q=head->next;
while(p!=NULL && q!=NULL)
{
        if(p==q)
        {
                //Loop detected!
                exit(0);
        }
        p=p->next;
        q=(q->next)?(q->next->next):q->next;
}
// No loop.
```

4.How do you find the middle of a linked list? Write a C program to return the middle of a linked list

```
// This function uses one slow and one fast
// pointer to get to the middle of the LL.
//
// The slow pointer is advanced only by one node
// and the fast pointer is advanced by two nodes!

void getTheMiddle(mynode *head)
{
        mynode *p = head;
        mynode *q = head;
        if(q!=NULL)
        {
                while((q->next)!=NULL && (q->next->next)!=NULL)
                {
                        p=(p!=(mynode *)NULL?p->next:(mynode *)NULL);
                        q=(q!=(mynode *)NULL?q->next:(mynode *)NULL);
                        q=(q!=(mynode *)NULL?q->next:(mynode *)NULL);
                }
                printf("The middle element is [%d]",p->value);
        }
}
```

5.If you are using C language to implement the heterogeneous linked list, what pointer type will you use?

The heterogeneous linked list contains different data types in its nodes and we need a link, pointer to connect them. It is not possible to use ordinary pointers for this. So we go for void pointer. Void pointer is capable of storing pointer to any type as it is a generic pointer type.

6.How to compare two linked lists? Write a C program to compare two linked lists.

Here is a simple C program to accomplish the same.

```
int compare_linked_lists(struct node *q, struct node *r)
{
        static int flag;
        if((q==NULL ) && (r==NULL))
        {
                flag=1;
        }
        else
        {
                if(q==NULL || r==NULL)
                {
                        flag=0;
                }
                if(q->data!=r->data)
                {
                        flag=0;
                }
                else
                {
                        compare_linked_lists(q->link,r->link);
                }
        }
        return(flag);
}
```

Another way is to do it on similar lines as strcmp() compares two strings, character by character (here each node is like a character).

7.How to create a copy of a linked list? Write a C program to create a copy of a linked list.

Check out this C program which creates an exact copy of a linked list.

```
copy_linked_lists(struct node *q, struct node **s)
{
        if(q!=NULL)
        {
                *s=malloc(sizeof(struct node));
                (*s)->data=q->data;
                (*s)->link=NULL;
                copy_linked_list(q->link, &((*s)->link));
        }
}
```

8.Write a C program to free the nodes of a linked list

Before looking at the answer, try writing a simple C program (with a for loop) to do this. Quite a few people get this wrong.
This is the wrong way to do it

```
struct list *listptr, *nextptr;
for(listptr = head; listptr != NULL; listptr = listptr->next)
{
        free(listptr);
}
```

If you are thinking why the above piece of code is wrong, note that once you free the listptr node, you cannot do something like listptr = listptr->next!. Since listptr is already freed, using it to get listptr->next is illegal and can cause unpredictable results!
This is the right way to do it

```
struct list *listptr, *nextptr;
for(listptr = head; listptr != NULL; listptr = nextptr)
{
        nextptr = listptr->next;
        free(listptr);
}
head = NULL;
```

After doing this, make sure you also set the head pointer to NULL!

9.Write a C program to return the nth node from the end of a linked list.

First, just keep on incrementing the first pointer (ptr1) till the number of increments cross n.
Now, start the second pointer (ptr2) and keep on incrementing it till the first pointer (ptr1) reaches the end of the LL.

```
ptr1 = head;
ptr2 = head;
count = 0;
while(count < n)
{
        count++;
        if((ptr1=ptr1->next)==NULL)
        {
                //Length of the linked list less than n. Error.
                return(NULL);
        }
}
while((ptr1=ptr1->next)!=NULL)
{
        ptr2=ptr2->next;
}
return(ptr2);
```

10.Write a C program to insert nodes into a linked list in a sorted fashion

The solution is to iterate down the list looking for the correct place to insert the new node. That could be the end of the list, or a point just before a node which is larger than the new node.
Note that we assume the memory for the new node has already been allocated and a pointer to that memory is being passed to this function.

```
void linkedListInsertSorted(struct node** headReference, struct node* newNode)
{
        // Special case for the head end
        if (*headReference == NULL || (*headReference)->data >= newNode->data)
        {
                newNode->next = *headReference;
                *headReference = newNode;
        }
        else
        {
                // Locate the node before which the insertion is to happen!
                struct node* current = *headReference;
                while (current->next!=NULL && current->next->data < newNode->data)
                {
                        current = current->next;
                }
                newNode->next = current->next;
                current->next = newNode;
        }
}
```

11.Write a C program to remove duplicates from a sorted linked list

As the linked list is sorted, we can start from the beginning of the list and compare adjacent nodes. When adjacent nodes are the same,remove the second one. There's a tricky case where the node after the next node needs to be noted before the deletion.

```
// Remove duplicates from a sorted list
void RemoveDuplicates(struct node* head)
{
        struct node* current = head;
        if (current == NULL)
                return;
        // do nothing if the list is empty
        // Compare current node with next node
        while(current->next!=NULL)
        {
                if (current->data == current->next->data)
                {
                        struct node* nextNext = current->next->next;
                        free(current->next);
                        current->next = nextNext;
                }
                else
                {
                        current = current->next; // only advance if no deletion
                }
        }
}
```

12.Write a program that reverses alternate elements in a given linked list

Input : a->b->c->d->e

Output : b->a->d->c->e.

```
Void  reverse_ two(Node **head)
{
        Node *p,*q,*r;
        p = *head;
        if (p->next)
        {
                q = p->next;
                *head = q;
        }
        else
                return;
        while(q->next && q->next->next)
        {
                r = q->next;
                p->next = q->next->next;
                q->next = p;
                q = p->next;
                p = r;
        }
        p->next = q->next;
        q->next = p;
        return;
}
```

13.Write a program to reverse a linked list using recursion

```
temp = recurse_reverse(head);
temp->next = NULL;

node * recurse_reverse(node *ptr)
{
        if(ptr->next = NULL)
        {
                head = ptr;
                return ptr;
        }
        node *temp = recurse_reverse(ptr->next);
        temp->next = ptr;
        return ptr;
}
```

14.Write a program to reverse a linked list without using recursion

```
Void reverse(node **head)
{
        Node  *q,*r,*s;
        q = *head;
        r = NULL;

        while(q)
        {
                S = r;
                R = q;
                Q = q->next;
                r->next = s;
        }

        *head = r;
}
```

15.Write a program to separate odd and even numbers in a linked list which are in sets of 3

```
node *oddeven(node *head)
```

```
{
	node *p,*q,*r,*s;
	r = head;
	if(r->next->next->next)
	{
		s = r->next->next->next;
		p = r;
		q = s;
	}
	else
		return r;
	while(q)
	{
		if(q->next && q->next->next && q->next->next->next)
		{
			p->next->next->next = q->next->next->next;
			p = q->next->next->next;
		}
		else
		{
			p->next->next->next = s;
			return r;
		}
		if(p->next && p->next->next && p->next->next->next)
		{
			q->next->next->next = p->next->next->next;
			q = p->next->next->next;
		}
		else
		{
			q->next->next->next = r;
			return s;
		}
	}

}
```

16.Write a program to sort a given linked list

```
void sortlist(node ** head)
{
	node *prev,*temp,*min,*r,*p,*q;
	int val;
	min = *head;
	r = *head;
	while(min != NULL)
	{
```

```
                val = min->data;
                temp = min->next;
                while(temp != NULL)
                {
                        while(val<temp->data)
                        {
                                prev = temp;
                                temp = temp->next;
                                if(temp == NULL)
                                        break;
                        }
                        if(temp!=NULL)
                        {
                                if(min->next == temp)
                                {
                                        min->next = temp->next;
                                        temp->next = min;
                                        if(min == (*head))
                                                *head = temp;
                                        else
                                                r->next = temp;
                                }
                                else
                                {
                                        p = temp->next;
                                        temp->next = min->next;
                                        prev->next = min;
                                        min->next = p;
                                        if(min == (*head))
                                                *head = temp;
                                        else
                                                r->next = temp;
                                }
                                q = temp;
                                temp = min;
                                min = q;
                                val = min->data;
                                prev = temp;
                                temp = temp->next;
                        }
                }
                r = min;
                min = min->next;
        }
}
```

17.Write a program to insert a number into a sorted double linked list

```
void sortlist(node **front,node **end,int num)
{
        node *temp;
        temp = new node;
        temp->data = num;
        if((*front) == NULL)
        {
                *front = *end = temp;
                temp->right = NULL;
                temp->left = NULL;
        }
        else
        {
                node *p,*q = *front;
                while(num > q->data)
                {
                        p = q;
                        q = q->right;
                        if(q == NULL)
                                break;
                }
                if(q == *front)
                {
                        temp->right = *front;
                        (*front)->left = temp;
                        temp->left = NULL;
                        *front = temp;
                        return;
                }
                if(!q)
                {
                        p->right = temp;
                        temp->right = NULL;
                        temp->left = p;
                        *end = temp;
                }
                else
                {
                        q->left->right = temp;
                        temp->right = q;
                        q->left = temp;
                        temp->left = p;
                }
        }
}
```

18.Write a program to sort a given double linked list

```
void sort_list(node **front,node **end)
{
	node *min,*temp,*p,*q,*s;
	int val;
	min = *front;
	while(min != NULL)
	{
		temp = min->right;
		val = min->data;
		while(temp != NULL)
		{
			while(val<temp->data)
			{
				temp = temp->right;
				if(temp == NULL)
					break;
			}
			if(temp)
			{
				if(min->right == temp)
				{
					if(temp->right)
						temp->right->left = min;
					else
						*end = min;
					if(min->left)
						min->left->right = temp;
					else
						*front = temp;
					min->right = temp->right;
					temp->right = min;
					temp->left = min->left;
					min->left = temp;
				}
				else
				{
					if(temp->right)
						temp->right->left = min;
					else
						*end = min;
					if(min->left)
						min->left->right = temp;
					else
						*front = temp;
					temp->left->right = min;
```

```
                                    min->right->left = temp;
                                    p = min->right;
                                    s = temp->left;
                                    temp->left = min->left;
                                    min->right = temp->right;
                                    temp->right =p;
                                    min->left = s;

                              }
                              //display(*front);
                              q = temp;
                              temp = min;
                              min = q;
                              val = min->data;
                              temp = temp->right;
                        }
                  }
                  min = min->right;
            }
      }
```