1. ***REVERSING OF THE LINKED LIST***

```
#include <stdio.h>

#include <stdlib.h>


struct Node {

   int data;

   struct Node* prev;

   struct Node* next;

};


struct Node* createNode(int data) {

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

   if (newNode == NULL) {

      printf("Memory allocation failed\n");

      exit(EXIT_FAILURE);

   }

   newNode->data = data;

   newNode->prev = NULL;

   newNode->next = NULL;

   return newNode;

}


void insertAtBeginning(struct Node** head, int data) {

   struct Node* newNode = createNode(data);
```

```
    if (*head == NULL) {
        *head = newNode;
    } else {
        newNode->next = *head;
        (*head)->prev = newNode;
        *head = newNode;
    }
}
void insertBeforeNode(struct Node** head, int key, int data) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* newNode = createNode(data);
    struct Node* current = *head;

    while (current) {
        if (current->data == key) {
            if (current->prev) {
                current->prev->next = newNode;
                newNode->prev = current->prev;
            } else {
                *head = newNode;
            }
            newNode->next = current;
            current->prev = newNode;
            return;
        }
        current = current->next;
    }

    printf("Key not found in the list\n");
}



void deleteNode(struct Node** head, int pos) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* current = *head;
    int count = 1;

    while (current && count < pos) {
        current = current->next;
        count++;
    }

    if (current == NULL) {
        printf("Position %d is beyond the length of the list\n", pos);
        return;
    }

    if (current->prev) {
        current->prev->next = current->next;
    } else {
        *head = current->next;
    }

    if (current->next) {
        current->next->prev = current->prev;
    }

    free(current);
    printf("Node at position %d deleted\n", pos);
}

void displayList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* current = head;
    while (current) {
        printf("%d-> ", current->data);
        current = current->next;
    }
    printf("\n");
}

void freeList(struct Node* head) {
    struct Node* current = head;
    struct Node* nextNode;

    while (current) {
        nextNode = current->next;
        free(current);
        current = nextNode;
    }
}

int main() {
    struct Node* head = NULL;
    int ch, newData, pos, key;

    while (1) {
        printf("\nMenu\n");
        printf("1. Insert at the beginning\n");
        printf("2. Insert before a node\n");
        printf("3. Delete a node\n");

        printf("4. Display list\n");

        printf("5. Free doubly linked list and exit\n");

        printf("Enter your choice: ");

        scanf("%d", &ch);


        switch (ch) {

          case 1:

            printf("Enter data to insert at the beginning: ");

            scanf("%d", &newData);

            insertAtBeginning(&head, newData);

            break;


          case 2:

            printf("Enter the value before which you want to insert: ");

            scanf("%d", &key);

            printf("Enter data to insert: ");

            scanf("%d", &newData);

            insertBeforeNode(&head, key, newData);

            break;


          case 3:

            printf("Enter the position you wish to delete: ");

            scanf("%d", &key);

            deleteNode(&head, key);

            break;
```

```
            case 4:
                printf("Doubly linked list: ");
                displayList(head);
                break;

            case 5:
                freeList(head);
                printf("Exiting the program\n");
                return 0;

            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

output:

```
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
```

```
Enter your choice: 1

Enter data to insert at the beginning: 25


Menu

1. Insert at the beginning

2. Insert before a node

3. Delete a node

4. Display list

5. Free doubly linked list and exit

Enter your choice: 1

Enter data to insert at the beginning: 26


Menu

1. Insert at the beginning

2. Insert before a node

3. Delete a node

4. Display list

5. Free doubly linked list and exit

Enter your choice: 1

Enter data to insert at the beginning: 27


Menu

1. Insert at the beginning

2. Insert before a node

3. Delete a node

4. Display list
```

```
5. Free doubly linked list and exit

Enter your choice: 4

Doubly linked list: 27-> 26-> 25->


Menu

1. Insert at the beginning

2. Insert before a node

3. Delete a node

4. Display list

5. Free doubly linked list and exit

Enter your choice: 2

Enter the value before which you want to insert: 3

Enter data to insert: 38

Key not found in the list


Menu

1. Insert at the beginning

2. Insert before a node

3. Delete a node

4. Display list

5. Free doubly linked list and exit

Enter your choice: 4

Doubly linked list: 27-> 26-> 25->


Menu

1. Insert at the beginning
```

```
2. Insert before a node

3. Delete a node

4. Display list

5. Free doubly linked list and exit

Enter your choice: 5

Exiting the program
```

2. <u>concation and saurting of the linked list</u>

```
#include <stdio.h>

#include <stdlib.h>


struct node

{

   int data;

   struct node *next;

};

void insertatbeg(struct node** head,int value)

{

   struct node* new_node=(struct node*)malloc(sizeof(struct node));

   new_node->data=value;

   new_node->next=*head;

   *head=new_node;

}

void concat(struct node *head1,struct node *head2)

{
   if (head1->next == NULL)
       head1->next = head2;
   else
       concat(head1->next,head2);
}
void sortlist(struct node** head1)
{
   struct node *temp,*i;
   for(temp=*head1;temp!=NULL;temp=temp->next)
   {
       for(i=temp->next;i!=NULL;i=i->next)
       {
           if(i->data < temp->data)
           {
               int tem=i->data;
               i->data=temp->data;
               temp->data=tem;
           }
       }
   }
}
void reverse(struct node** head1)
{
   struct node *prev=NULL;
   struct node *current=*head1;
   struct node* next=NULL;
   while(current!=NULL)
   {
       next=current->next;
       current->next=prev;
       prev=current;
       current=next;
   }
   *head1=prev;
}
void printlist(struct node* node)
{
   struct node* temp=node;
   while(temp!=NULL)
   {
       printf("%d-->",temp->data);
       temp=temp->next;
   }
   printf("NULL\n");
}
int main()
{
   struct node *head1=NULL;

   insertatbeg(&head1,10);
   insertatbeg(&head1,40);
   insertatbeg(&head1,20);
   insertatbeg(&head1,50);
   printf("List 1:");
   printlist(head1);

   struct node *head2=NULL;
   insertatbeg(&head2,50);
   insertatbeg(&head2,70);
   insertatbeg(&head2,60);

   printf("List 2:");
   printlist(head2);

   concat(head1,head2);
   printf("List after concatenation:");
   printlist(head1);

   sortlist(&head1);
   printf("List after sorting:");
   printlist(head1);

   reverse(&head1);
   printf("Reversed Linked list");
   printlist(head1);
}
```

output:

```
List 1:50-->20-->40-->10-->NULL

List 2:60-->70-->50-->NULL

List after concatenation:50-->20-->40-->10-->60-->70-->50-->NULL

List after sorting:10-->20-->40-->50-->50-->60-->70-->NULL

Reversed Linked list70-->60-->50-->50-->40-->20-->10-->NULL
```