Siudhu Sankati
AP19110010477

```
// llist.h header file - custom.
/* library for input & output operations of a linked list */
#include <stdio.h>
#include<stdlib.h>
typedef struct Node{
    int data;
    struct Node *next;
};
Node* createNode(int data){
    node *n = (cnode*) malloc (sizeof(node));
    n->data = data;
    n->next = NULL;
    return n;
}
node* createList(){
    int n, data;
    node *p, *head = NULL;
    printf("\n How many elements to enter");
    scanf("%d", &data);
    while(n--){
        printf("Enter number");
        scanf("%d", &data);
        if (head == NULL){
            head = createNode(data);
            p = head;
        }
    else{
```

```
        p->next = createNode(data);
        p = p->next;
        }
    }
    return head;
}
void display(node *head){
    while(head != NULL){
        printf("%d->", head->data);
        head = head->next;
    }
    printf("NULL\n");
}
```

#

1. Write a program to insert and delete an element at the nth and kth position in a linked list where n and k is taken from user.

```
#include<stdio.h>
#include<stdlib.h>
#include "lli.h"
node* insertAtPos(node *head, int pos, int data){
    if (pos == 0){
        node* newNode = createNode(data);
        newNode->next = head;
        return NewNode;
    }
    if (head == NULL){
        printf("error\n");
        return head;
    }
    head->next = insertAtPos(head->next, pos-1, data);
    return head;
}

node* deleteAtPos(node *head, int pos){
    if (pos == 0){
        return head->next;
    }
    if (head == NULL){
        printf("error\n");
```

```
        return head;
    }
    head -> next = deleteAtPos(head->next, pos-1);
    return data;
}

int main()
{
    int data, n, k;
    node * head = createList()
    printf("\n enter n, index of where you want to add a
            node");
    scanf("%d", &n);
    printf("\n enter data of the node");
    scanf("%d", &data);
    head = insertAtPos(head, n, data);
    printf("\n after entering, list looks like \n");
    display(head);

    printf("\n enter k, index of where you want to delete
            a node \n");
    scanf("%d", &k);
    head = deleteAtPos(head, k);
    printf("\n after deleting, list looks like \n");
    display(head);

    return 0;
}
```

Output

How many elements to enter? 5

Enter a number

1

Enter a number

2

Enter a number

3

Enter a number

4

Enter a number

6

enter n, index of where you want to add a node 4

enter data of the node 5

after entering, list looks like

1 -> 2 -> 3 -> 4 -> 5 -> 6 -> NULL

enter K, index of where you want to delete a node

2

after deleting, list looks like

1 → 2 → 4 → 5 → 6 → NULL.

2. Construct a new linked list by merging alternate nodes of two lists for example in list1 we have have {1,2,3} and in list2 we have {4,5,6} in the new list we should have {1,4,2,5,3,6}.

```
#include "iio.h"

struct node* joinAltCnode *list1, node *list2){
    node *newlist = NULL, *tmp;
    if (list1 == NULL && list2 = NULL))
        return NULL;
    if (list 1 == NULL)
        return list2;
    if (list2 == NULL)
        return list1;
    newlist = list 1;
    tmp = list1;
    list1 = list1->next
    int i = 0;
    while(1){
        if (i%.2 != 0 && list1 != NULL){
            tmp->next = list2;
            list1 = list1->next;
        }
        else if (list2 != NULL){
            tmp->next = list2;
            list2 = list2->next;
        }
```

```
        else
            break;
        }
        tmp = tmp->next;
        i++;
    }
    return newlist;
}
int main(){
    node * list1 = createList(), * list2 = createList();
    printf("\n list1 was \n");
    display(list1);
    printf("\n list2 was \n");
    display(list2);

    node *newlist = joinAlt(list1, list2);
    printf("\n after merging lists: ");
    display(newlist);

    return 0;
}
```

Output.

```
How many elements to enter? 1
Enter a number
1
How many elements to enter? 2
Enter a number
2
Enter a number
3
list 1 was
1 -> NULL
list 2 was
2 -> 3 -> NULL
after merging lists 1->2->3->NULL
```

3. Find all the elements in the stack whose sum is equal to k (value k is given from user)

```
# include <stdio.h>
# include < stdlib.h>
# include < limits.h>
# define max 1000

typedef struct STACK{
        int ar[max];
        int top;
}
void push (stack *s, int data){
    if (s->top >= max-1){
            exit(0);
        }
        s->top++;
        s->ar [s->top] = data;
}
int pop (stack *s){
        if (s->top < 0) return INT_MIN;
        int temp = s->ar[s->top];
        s->top--;
        return temp;
}
```

```
void display(stack s){
    int i;
    for(i = s.top; i>-1; i--){
        printf("%d ", s.ar[i]);
    }
    printf("\n");
}

void sumk(stack s1, stack v, int k){
    if(k==0){
        display(v);
        return;
    }
    if(s1.top == -1) return;
    int temp = pop(&s1);
    sumk(s1,v,k);
    stack v1 = v;
    push(&v1, temp);
    sumk(s1, v1, k-temp);
}

int main(){
    stack arr, v;
    arr.top = -1;
    v.top = -1;
```

```
int expected, n, num;
printf("enter the number of element you wan to
        the stack\n");
scanf("%d", &n);
while(n--){
    printf("number\n");
    scanf("%d", &num);
    push(&arr, num);
}
printf("enter the expected sum\n");
scanf("%d", &expected);
n = arr.top+1;
sumk(arr, n, expected);
return 0;
}
```

Output

```
Enter the number of element you want in the stack.
10
number
1
number
2
number
3
number
7
```

number

8

number

9

number

4

number

5

number

5

number

6

Enter expected value

15

7 8

1 2 3 9

1 2 3 8 1

2 3 9 1

1 2 7 5

3 7 5

2 8 5

1 9 5

2 7 15

1 8 15

9 15

7 8

1 2 3 1 8

2 5 8

1 1 5 8

2 7 6

1 8 6

9 6

1 2 1 6

8 8 6

1 3 5 6

1 2 1 5 6

3 1 5 6

1 8 6

1 8 6

3. Write a program to input print the elements in a queue.

i. in reverse order

ii. in alternate order.

```
# include <stdio.h>
# include <stdlib.h>

typedef struct Node{
        int data;
        struct Node * next;
} node;

typedef struct Queue{
        node *front, *rear;
} queue;

node* newNode (int k){
    node* temp = (node*) malloc (sizeof (node));
    temp -> data = k;
    temp -> next = NULL;
return temp;
}
```

```
queue createQueue(){
    queue q;
    q.front = q.rear = NULL
    return q;
}

void enqueue(queue *q, int k){
    node * temp = newNode(k);
    if (q->rear == NULL){
        q->front = q->rear = temp;
        return;
    }
    q->rear->next = temp;
    q->rear = temp;
}

void displayAlt(queue q){
    while (q.front != NULL){
        printf("%d ->", q.front->data);
        if (q.front->next != NULL) q.front = q.front->next->next;
        else break;
    }
    printf("NULL\n");
}
```

```
void displayRev(queue q){
    if (q.front == NULL){
        printf("NULL");
        return;
    }
    int temp = q.front -> data;
    q.front = q.front -> next;
    displayRev(q);
    printf("<- %d", temp);
}
int main(){
    queue q = createQueue();
    int n, num;
    printf("enter the number of element you want in
            the queue \n");
    scanf("%d", &n);
    while(n--){
        printf("number\n");
        scanf("%d", &num);
        enQueue(q, num);
    }
    displayRev(q);
    printf("\n");
```

```
    displayAlt(q);
    return 0;
}
```

Output:

```
Enter the number of element you want in queue
8
number
1
number
2
number
3
number
4
number
5
number
23
number
1
number
24
```

NULL ← 24 ← 1 ← 23 ← 5 ← 4 ← 3 ← 2 ← 1

1 → → 3 → → 5 → → 1 → → NULL

5. i) How array is different from linked list.

arrays	linkd list
Fixed size: Resizing is expensive	Dynamic size.
Insertions and Deletions are inefficents. elements are usually shifted	Insertion and deletions are efficent: No shifting.
Random access ie efficent indexing	No random access → Not suitable for operations requiring accessing elements by index such as sorting.
No memory waste if the array is full or almost full; otherwise may result in much memory waste	since memory is allocated dynamically no memory waste.
sequential access is faster	Sequential access is slow.

(ii) Write a program to add the first element of one list to another list for example we hae {1,2,3} in list1 and {4,5,6} in list2 we have to get {4,1,2,3} as output for list1 and {5,6} for list2.

```
# include "iio.h"

void shiftFirstElement (node **list1, node **list2){

    if (*list2 ==NULL) return;
```

```
    node *temp = *list2
    *list2 = temp->next;
    temp->next = *list1;
    *list1 = temp;
}

int main(){
    struct node *list1 = createList(), list2 = createList();
    shiftFirstElement(&list1, &list2);
    printf("\n list1 is: ");
    display(list1);
    printf("\n list2 is ");
    display(list2);
}
```

Output

```
How many elements to enter? 4
Enter a number
1
Enter a number
2
Enter a number
3
Enter a number
4
Enter a number
5
```

How many elements to enter ?3

Enter a number

1

Enter a number

2

Enter a number

3

List 1 is : 1 → 1 → 2 → 3 → 4 → NULL

List 2 is : 2 → 3 → NULL.