Assignment - (C DSA)

P. Vishal chowdary
AP19110010056

1. Program to insert and delete an element at Nth and kth position

```
#include <stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node * next;
}
display(struct node * head)
{
if (head == Null)
{
printf ("Null \n" );
}
else
{
printf ("%d \n", head → data );
display(head → next);
}
}
del(struct node* before_del )
{
struct node * temp;
temp = before_del → next;
before.del → next = temp → next;
free (temp);
}
Struct node* front (struct node * head, int value)
{
struct node* p;
p = malloc (size of (struct node));
p → data = value;
p → next = head;
return (p);
}
end (struct node * head, int value)
{
struct node * p, * q;
p = malloc (size of (struct node));
```

```
p->data= value;
p-> next = Null;
 q=head;
 while (q->next != NULL)
{
 q= q-> next;
}
 q -> next= p;
}
 after(struct node*  a ;int value)
 {
  if(a->next != NULL)
 {
 Struct node *p;
 p= malloc (size of (structnode));
 p -> data =value;
 p -> next =a-> next;
  a -> next = p;
 }
 else
 {
  printf("USE END FUNCTION TO INSERT AT THE END\n");
 }
 }
 int main( )
 {
 struct node *prev, *head ,*p ;
 int a,i ;
 Printf ("NUMBER OF ELEMENTS")
  Scanf ("%d", & a);
  head = Null;
  for(i= 0;i<a;i++)
  {
  P= malloc (size of (Struct node));
  Scanf ("%d", & P-> data);
  P-> next = NULL;
  if (head-> NULL)
   head = P;
```

```
else
prev->next=p;
  prev=p;
}
head=front(head,10);
 end(head,20);
  after(head->next->next,30);
  del(head->next);
   del(head->next->next);
   display(head);
    return 0;
 }
```

OUTPUT

```
Number of elements 5
  1
  2
  3
   4
   5
   10
    1
   30
    4
    5
    20
    NULL
```

2. New Linked list By Merging Alternate Nodes

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
int data;
struct Node* next;
};
void push (struct Node** head_ref, int new_data)
{
struct Node* new_node = (struct Node*) malloc (size of (struct Node
  new_node -> data = new_data;
  new_node -> next = (*head_ref);
  (*head_ref) = new_node;
  }
  void printlist(struct Node * head)
  {
  struct Node* temp = head;
  while (temp != NULL)
  {
  printf("%d ", temp->data);
    temp = temp -> next;
   }
   printf("\n");
   }
  void merge(struct Node *p , struct Node** q)
  {
  struct Node * p_curr = p; *q_curr = *q;
  struct Node * p_next, * q_next;
  while (p_curr != NULL && q_curr != NULL)
  p_Next = p_curr -> next;
  q_next = q_curr -> next;
  q_curr -> next = p_next;
  p_curr -> next = q_curr;
  p_curr = p_next;
  q_curr = q_next
  }
```

```
*q = q->curr;
}
int main()
{
struct Node *P = NULL, *q = NULL;
Push(&P, 2);
Push(&P, 1);
Push(&P, 3);
Printf("1st LINKED LIST\n");
Printlist(P);

Push(&q, 3);
Push(&q, 8);
Push(&q, 6);
Printf("2nd Linked List\n");
Printlist(q);
merge(P, &q);
Printf("CHANGED LINKED LIST\n");
Print list(P);
return 0;
}
```

output

1st linked list

3 2 1 3 4 2

2nd linked List

6 8 3

changed LINKED LIST

3 6 4 8 2 3

3 6 4 8 23

3. Find all the elements in the stack whose sum is equal to k (where 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;
} Stack;

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 r,int k){
  if(k==0){
   display(r);
   return;
}
if(s1.top==-1) return;
int temp=pop(&s1);
sumk(s1,r,k);
```

```
stack v1 = v;
push(&v1, temp);
sumk(s1, v1, k-temp);
}
int main(int argc, char const *argv[]) {
  stack arr, v;
  arr.top = -1;
  v.top = -1;
  int expected, n, num;
  printf("enter the number of element you want in stack");
  scanf("%d", &n);
  while(n--) {
  printf("number\n");
  scanf("%d", &num);
  push(&arr, num);
}
printf("enter expected value\n");
scanf("%d", &expected);
n = arr.top+1;
sumk(arr, v, expected);
return 0;
}
```

4. Write a program to print the elements in a queue.

(a) In reverse order

(b) In alternate order

```
#include<stdio.h>
#include<stdlib.h>
typedef struct Node{
  int data;
  struct Node *next;
}node;
typedf struct queue{
   node *front, *rear;
}queue;
node* newNode(int k)
{
  node* temp=
   (node*) malloc (size of (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;
}
```

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

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

Array	Linked list
1. Size of an array is fixed	1. Size of a list is not fixed
2. It occupies less memory than a linked list for the same number of elements	2. It occupies more memory
3. Deleting an element from an array is not ~~responsible~~	3. Deleting an element is possible
4. Insertion and deletion take more time.	4. Insertion and deletion process take less time

```
#include<stdio.h>
#include <stdlib.h>
  struct Node
{
int data;
struct Node* next;
};
void push(struct Node** head_ref, int, new_data)
{
struct Node* new_node = (struct Node*)malloc(size of (struct node));
new_node -> data = new_data;
new_node -> next = (* head_ref);
(* head_ref) = new_node;
}
void printlist (struct Node *head)
{
  struct Node * temp = head;
  while (temp != NULL)
{
printf("%d", temp -> data);
  temp = temp -> next;
```

```
}
printf("\n");
}
void merge(struct Node *p, struct Node **q)
{
{
struct Node * p_curr = p, * q_curr = *q;
struct Node *p_next, *q_next;
while (p_curr != Null && q_curr != NULL)
p_next = p_curr -> next;
q_next = q_curr -> next;
q_curr -> next = q_curr;
p_curr = p_next;
q_curr = q_next;
}
*q = q_curr;
}
int main()
{
struct Node * p = NULL, *q = NULL;
push(&p, 6);
push(&p, 8);
push(&p, 9);
printf("1st LINKED LIST\n");
printlist(p);
push(&q, 7);
push(&q, 9);
push(&q, 3);
push(&q, 8);
push(&q, 6);
printf("2nd LINKED LIST\n");
printlist(q);
merge(p, &q);
printf("CHANGED 1st LINKED LIST\n");
printlist(p);
printf("CHANGED 2nd LINKED LIST\n");
printlist(q);
```

output

1st ~~link~~:

1st LINKED LIST

9 8 6

2nd LINKED LIST

6 8 3 9 7

CHANGED 1st LINKED LIST

9 ~~8~~6 8 8 6 3

CHANGED 2nd LINKED LIST

9 7