Asritha Meka
AP19110010224

Assignment - 04.

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

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define initMemory() ((struct node*)
malloc (sizeof (struct node)))
struct node
{
    int data;
    struct node *next;
};
struct node*createlist()
{
 int i, n;
 struct node *newhead, *head = NULL;
printf ("\n how many elements to enter-?");
 scanf("%d", &n);
  if (n==0)
  {
  return head;
  }
  for (i=0; i<n; i++)
  {
     if (i==0)
```

```
{
  Head = initMemory();
    Newhead = Head;
  }
else
{
    New Head->next = initMemory();
  Newhead=>next; New head->next;
  }
  printf("enter the %dth element", i);
   scanf("%d", &new head->data);
 }
newhead->next = NULL;
  return (Head);
}
void display(struct node *head)
{
  if (head==NULL)
  {
   printf("no element in the list");
     return;
  }
  struct node * ptr = head;
  while (ptr != NULL)
   printf("%d", ptr->data);
```

```
        ptr=ptr->next;
      }
    }
void insertatN(struct node **head, int n)
{
  int i;
  struct node *newptr, *ptr, *tmp;
  newptr=initMemory();
  printf("enter the element to be
    inserted");
  scanf("%d", &newptr->data);
  if (n==0)
    {
    newptr->next = *head;
    *head = newptr;
    return;
    }
  ptr = *head;
  for(i=0; i<n-1; i++)
  {
    if (ptr->next==NULL)
    {
      if(i!=n-1)
      {
      printf("List is not initialized
till n\n entering the new node
  at end\n");
```

```
        }
        ptr->next = newptr;
        newptr->next = NULL;
        return;
    }
    else
    {
      ptr = ptr->next;
    }
}
temp = ptr->next;
ptr->next = newptr;
newptr->next = tmp;
}
void deleteAtk(struct node **head,
  int k)
{
  int i;
struct node *tmp;
 if(k==0)
 {
    printf("\n %d is deleted\n",
        (*head)->data);
    tmp = *head;
  *head = (*head)->next;
  free(tmp);
```

```
    return;
  }
  struct node *ptr = *head;
  for (i=0; i<k-1; i++)
  {
    if (ptr->next == NULL)
    {
      printf("\n no element in pos:%d\n",
                                    k);
        return;
      }
    else
      {
        ptr = ptr->next;
      }
    }
    tmp = ptr->next;
    printf("%d is deleted in", tmp->
                          data);
    ptr->next = tmp->next;
    free(tmp);
}
int main()
{
  int n, k;
  struct node *head = createlist();
  printf("\n enter n index of where
you want to add a node");
```

```
    display(head);
    printf("\nenter k, index of where
    you want to delete a node\n");
    Scanf("%d", &k);
    deleteAtk(&head, k);
    Prunt("\n after deleting, list
     looks like\n");
     display(head);
     return 0;
}
```

---

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

code:

```
# include <stdio.h>
# include <stdlib.h>
# define initMemory() ((struct node*)
malloc(sizeof(struct node)))
struct node
{
    int data;
    struct node *next;
};
```

```
void printlist (structnode *head)
{
 structnode* ptr=head;
  while (ptr)
   {
   printf("%d→", ptr->data);
     ptr=ptr→next; }
     printf("NULL)n");
   }
  void push (struct node *head, int data)
 {
struct node *new=(struct node*)malloc
                  (sizeof(struct node));
new→ data = data;
 new→ next = *head;
* head =new;
}
 struct node *merge(struct node *a,
   struct node*merge(sb)
 {
    struct node dummy;
     struct node*fail= dummy;
   dumny.next = Null;
    while(1) {
   if(a==NULL)
   {
    tail→next=b;
      break;
```

```
    }
    else if (b==NULL)
    {
     tail->next=a;
     break;
    }
    else
    {
     tail->next=a;
     tail=a;
     a=a->next;
        tail->next=b;
    }
    }
    return dummy.next;
    }
    void main()
    {
    int keys[]={1,2,3,4,5,6,7};
    int n=sizeof(keys)/sizeof key[0];
    struct node*a=NULL,*b=NULL;
    for(int i=n-1; i>0; i=i-2)
       push(&a, keys[i]);
    for (int i=n-2; i>=0; i=i-2)
     push(&b, key(i));
    struct node* head=merge(a,b);
     print list(head);
```

Scanned with CamScanner

(3) find all the elements in the stake whose sum is equal to k.

coding

```
#include <stdio.h>
int top=-1;
int x;
char stack[100];
void push(int x);
char pop();
int main()
{
int i,n,a,t,k,f,sum=0, count=1;
printf("enter the number of elements
      in the stack");
scanf("%d",&n);
for(i=0;i<n;i++){
printf("enter next element");
scanf("%d",&a);
push(a);
}
printf("enter the sum to be checked");
scanf("%d",&k);
for(i=0;i<n;i++)
{
t=pop();
sum+=t;
```

```
count++;
if (sum==k){
for (int j=0;j<count;j++)
printf("%d",stack[j]);
f=1
break;
}
push(t);
}
if(f!=1)
printf("The elements in the stack dont
add up to sum");
}
void push(int x)
{
if(top==99)
{
printf("\n stack is FULL!!!\n");
return;
}
top=top+1;
stack[top]=x;
}
char pop()
{
if(stack[top]==-1)
{
printf("\n stack is EMPTY!!!\n");
```

```
return 0;
}
x = stack[top];
top = top - 1;
    return x;
}
```

---

(4) write a program to print the elements in a queue.
(i) in reverse order (ii) in alternate order.

coding:-

```
#include<stdio.h>
#include<stdlib.h>
struct node.
{
  int data;
    struct node* next;
  };
  void print rev(struct node* head)
  {
  if (head == NULL)
   return;
     printf rev(head -> next);
     printf("%d", head -> data);
void push(struct node* headrev, char new)
{
 struct node* node_new = (struct node*)
                  malloc (size of (struct
                        node));
```

```
node_new->data = new;
node_new->next = (*head_ref);
(*head_ref) = node_new;
}
int main()
struct node* head = NULL;
push(&head, 4);
push(&head, 3);
push(&head, 2);
print new(head); print alternate
(head);
return 0;
}
void print alternate(struct node* head)
{
int count = 0;
while(head != NULL)
{ if (count %2 == 0)
    count << head -> data << " ";
    count++;
    head = head -> next;
}
```

5 (a) (i) How array is different from Linked list.

Arrays:-

(i) fixed size: Resizing is expensive.

(ii) Insertion and deletions are inefficient. elements are usually shifted.

(iii) No memory waste if the array is full (or) almost full. otherwise may result in much memory waste.

Linked list:-

(i) Dynamise size.

(ii) Insertions and deletions are efficient. No shifting.

(iii) Since memory is allocated dynamically.

(b) (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.

coding:-

```
#include <stdio.h>
#include <stdlib.h>
int len(int a[])
{
int i=0, a=0;
```

```
while (1)
{
 if (a[i])
 {
   an++, i++;
 }
else
 {
 break;
  }
 }
  return an;
}
void changing list (int a[a], int b[7]
{
  for (int i= len(a)-1; i>=0; i--)
  {
  a[i+1] = a[i];
  }
  a[0] = b[0];
  printf("\n the elements of first array
        :\n");
for(int i=0; i<len(9); i++)
{
 printf("%d", a[i]);
```

```
}
for(int i=0; i<len(b); i++)
  {
    b[i] = b[i+1]; }
printf("\n the elements of second
        array:\n");
for(int i=0; i<len(b); i++)
{printf("%d", b[i]);
}
}
int main()
{
 int a[10] = {1,2,3}, b[10] = {4,5,6};
 changing list(a,b);
}
```