G. krishna vamsi
AP19110010356
CSE-H.

1(i) #include <stdio.h>
#include <stdlib.>

```
struct johny{
} *pj
struct Node{
  int data;
  struct Node *next;
 };
struct Node* head;
void print()
{
 struct Node* temp = head;
 while(temp != NULL)
  {
 printf("%d", temp->data);
 temp = temp-> next;
}
printf("\n");
}
void insert(int date, int n)
{
 int i;
 struct Node* temp1 = (struct Node*) malloc (sizeof (struct Node));
 temp1-> data = data;
 temp1->Next = Nutll;
 if (n == 1)
{
 temp1->nemt=head;
  head = temp1;
  return;
```

```
struct Node* temp2 = head;
for (i=0; i<n-2; i++)
{
 temp2 = temp2 -> next;
 }
 temp1 -> next = temp2 -> next;
 }
 temp1 -> next = temp2 -> next;
 temp -> next = temp1;
}
int main ()
{
 head = NULL;
 Insert (2,1);
 insert (3,2);
 insert (4,1);
 insert (5,2);
 print();

 return 0;
 }
```

output :-

4 5 2 3

---- program finished with exit code 0

press ENTER to exit console.

(1(ii)) //Linked list: Delete a node at nth position.

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

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

struct Node* head;

void insert(int data); // insert and integer at the end of list
void print(); // print all elements in the list.
void Delete(int n); delete node at position n.

int main()
{
head = NULL; // list is first empty
insert(2);
 insert(4);
 insert(6);
 insert(5); // list 2 4 6 5
 printf("Enter a position:");
 scanf("%d", &n);
 Delete(n);
 print();
 return 0;
}
```

```
struct Node* temp = head;
while(temp != NULL)
{
printf("%d", temp->data);
temp = temp->next;
}
}
```

```
void Delete(int n)
{
temp->next = NULL;
 struct Node* temp2 = head;
 while(1)
 {
 if(temp2->next == NULL)
 {
 temp2->next = temp;
 break;
 }
  temp = temp2->next;
 }
}
void print()
```

```
void delete (n)
{
 int i;
 struct Node* temp1 = head;
 if (n == 1) {
   head = temp1->next; // head points to second node.
   free (temp1);
   return;
 }
 for (i=0; i<n-2; i++)
 {
   temp1 = temp1->next;
 }
 // temp1 points to (n-1)th node.
 struct Node* temp2 = temp1->next; // nth node.
 temp1->next = temp2->next; //(n+1)th node.
 free (temp2);
}

void Insert (int n)
{
 int i;
 struct Node* temp = (struct Node*) Malloc (sizeof (struct
 temp->data = n;
 if (head
```

```
temp->next = NULL;
struct Node* temp2 = head;
while(1)
{
if(temp2->next == NULL)
{
temp2->next = temp;
break;
}
temp = temp2->next;
}
void print()
{
```

```
struct node *tempobet = head;
while(tempobet != NULL)
{
printf("%d", tempobet->data);
tempobet = tempobet->next;
}
}
```

(2)

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
  int data;
  struct Node *next;
};
void push(struct Node ** head_ref, int new_data)
{
  struct Nod* New_node =
        (struct Node*) malloc (sizeof (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,3);
  push(&p,2);
  push(&p,1);
  printf("first linked list:\n");
  printList(p);
 push(&q,8);
 push(&q,7);
 push(&q,6);
 push(&q,5);
 push(&q,4);
 printf("second linked list:\n");
 printList(p);
 printf("Modified first linked list:\n");
 printList(p);
 printf("Modified second linked list:\n");
 printList(q);
 getchar();
 return 0;
}
```

output:-

```
Modified first linked list:
1 4 2 5 3 6
Modified second linked list:
7 8 . .
```

```
#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 a 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 += 1;
if(sum == k){
for(int j=0; j<count; j++)
printf("%d", stack[j]);
f = 1;
break;
}
push(t);
}
```

```
push(t);
}
if(fl=1)
printf("The elements in the stack done add up to the sum");
}
void push(int x)
{
if(top==99)
{
printf("\nstack is FULL!!!\n");
return;
}
top=top+1;
stack[top]=x;
}
char pop()
{
if(stack[top]==-1)
{
printf("\nstack is empty!!!\n");
return 0;
}
x=stack[top];
top=top-1;
return x;
}
```

output:

```
Enter the number of elements in the stack
Enter next element
Enter the sum to be checked.
```

4e)

```
// Function to reverse the queue.
void reverseQueue(queue<int>& Queue)
{
 stack<int> stack;
 while (!queue.empty()) {
  stack.push(queue.front());
  Queue: pop();
}
while (!.stack, empty()) {
    Queue.push(stack.top());
    stack.pop();
  }
}
```

output;

1
4
1
8
1
3

} Queue

output;

1
4
1
8
1
3

} stack.

4(ii)

```
void reverseQueueFirstKelements(int k, queue<int> &Queue)
{
  if (Queue.empty() == true || k > Queue.size())
     return;
  if (k <= 0)
     return;
  stack<int> stack;
  /* push the first k elements into a stack */
  for (int i=0; i<k; i++) {
    stack.push(Queue.front());
    Queue.pop();
  }
  while (!stack.empty()) {
    Queue.push(stack.top());
    stack.pop();
  }
  for (int i=0; i<Queue.size()-k; i++) {
    Queue.push(Queue.());
    Queue.pop();
  }
}
```

Output :-

5
4
3
6
1
2