DSA

Name :- E. HemaSundar
Roll :- AP19110010481.

1) Ans :-

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
int data ;
struct Node * next ;
} ;
Struct Node *head ;
void Insert (int data ; int n) {
Node * temp = new node () ;
temp -> next = Null ;
temp -> data = data ;
iF (n = = 1) {
temp -> next = head ;
head = temp ;
return ;
}
```

```
void Delete -(int k) {
Struct Node * temp = head ;
if (k = = 1){
head = temp → next ;
Free (temp);
Return ;
}
Node * temp =head ;
For (int i=0 ; i< n-2 ; i++) {
temp = temp → next ;
}
temp → next = temp → next ;
temp → next = temp ;
}
void print ( ) ;
For (int i=0 ; i< k-2 ; i++)
temp = temp → next ;
Free (temp) ;
}
int main ( ) {
int n, x, k ;
head = NULL ;
PrintF ("Enter the position for and inserting :");
Scanf ("%d", &n) ;
Scanf ("%d", &x) ;
Insert (x, n);
PrintF ("Enter the position to delete");
Scanf("%d", &k);
delete (F);
```

```
Print (x) ;
return ;
}
```

2) Ans :-

```
#include <stdio.h>
#include <stdlib.h>
#include <assertin>
Struct . node
{
 int data ;
 Struct node * next ;
};
void move node (Struct node **x ; Struct node **y );
Struct node * sorted merge (Struct node* a, Struct node* b)
{
Struct node dummy ;
Struct node * tail = &dummy ;
dummy . next = Null ;
while (1)
{
 if (a == Null)
 {
  * y = new node -> next ;
   new node -> next = * x ;
   * x = new node ;
 }
```

```
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 point list (struct node * node)
{
while (node != Null)
{
printf (" %d ", node -> data);
node = node -> next;
}
}
tail -> next = b;
break;
}
else if (b == Null)
{
tail -> next = a;
break;
}
if (a -> data <= b -> data)
{
move node (&(tail -> next), &a);
}
tail = tail -> next;
}
return (dummy.next);
}
```

```
void move.node =(struct node ** x, struct node **y)
{
 struct node* newnode = *y;
 assert (newnode != NULL);
 int main()
 {
  struct node * res = null;
  struct node * a = null;
  struct node * b = null;
   Push (&a, 1);
   Push (&a, 2);
   Push (&a, 3);
   Push (&b, 4);
   Push (&b, 5);
   Push (&b, 6);
   res = sorted merge (a, b);
   printf (" merge linked list is : \n");
   Printf list (res);
   return 0;
   }
```

3) Ans:

```
#include<stdio.h>
int S1[10], top1 = -1, S2[10], top2 = -1;
int S1 empty()
{
if (top==-1)
return 1;
else
  return 0;
}
int S1 top()
{
 return S1 [top1];
}
int S1 pop()
{
  top1--;
}
int S1 push (int x)
{
 S1 [++top1]=x;
}
 int S2 empty()
{
 if (top2 == -1)
    return 1;
 else
    return 0;
 }
```

```
int S2 Top()
{
 return S2[top2];
}
int S2 pop()
{
 top2--;
}
int S2 Push (int x)
{
S2[++ top2] = x;
}
int sum (int k)
{
 int x;
 while (S1 empty ()!=1)
 {
  x = S1 TOP();
  S1 POP();
  while (S1 empty () != 1)
  {
   if (x + S1 top () = k)
   {
    Printf ("%d, %d) \n", x, S1 top ());
    }
    S2 Push (S1 TOP());
    S1 POP ();
  }
  while (S2 empty () != 1)
  {
  S1 Push (S2 = TOP());
```

```
        S2 Pop();
    }
  }
}

int main()
{
  int n, i, e, k;
  printf("Enter the no. of elements of stack: \n");
  scanf("%d", &n);
  for(i=0 ; i<n ; i++)
  {
   scanf("%d", &e);
   S1 Push(e);
  }
  printf("Enter the value of constant sum: \n");
  scanf("%d", &k);
  printf("The combinations whose sum is equal
                    to %d : \n");
   Sum(k);
  }
```

4) Ans :-

```
#include <stdio.h>
# define size 10
void insert (int);
void delete ();
int queue[10], F= -1, R= -1;
void main()
{
int value, choice;
while (1){
printf ("\n\n *** menu *** \n");
printf ("1. insertion \n 2. Deletion \n 3. Print reverse \n
        4. Print alternate \n 5. exit");
printf ("\n Enter your choice");
scanf ("%d", &choice);
switch (choice){
    case 1:
      printf ("Enter the value to insert ");
      scanf ("%d", &value);
      insert (value);
      break;
      case 2;
      delete ();
      break;
      case 3;
       printf ("The Reversed queue is ");
       for (int i= size; i>=0; i--)
{
```

```
if (queue[i] == 0)
continue;
Printf("%d ", queue[i]);
}
break;
case 4:
    Printf("Alternate elements of queue are");
    For(int i=0; i<size; i=2)
{
  if (queue[i] == 0)
  continue;
  Printf("%d ", queue[i]);
}
  break;
  case 5:
     exit(0);
  default:
     Printf("\n wrong selection");
  }
}}

void insert (int value){

    if ((F==0 && n==size-1) || F==n+1)

      Printf("\n Queue is Full and insertion can't be
                                    done");
    else {
      if (F == -1)

  F=0;
      n= (n+1) %size;
      queue[n] = value;
```

```
        printf("\n Insertion success");
    }
}

void delete() {

    if (f == -1)
        printf("\n Queue is empty and deletion
                can't be done");
    else {
        printf("\n Deleted %d", queue[f]);
        f = (f+1) % size;
        if (f == n)
            f = n = -1;
    }
}
```

5) Ans :-

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

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

void printlist(struct node* head)
{
    struct node * ptr = head;
    while(ptr)
    {
        printf("%d", ptr->data);
        ptr = ptr->next;
    }
```

```
void push (struct node ** head , int data)
{
  struct node * new node = (struct node*) * malloc
                          (size of struct node) ;
   new node -> data =data ;
   new hode -> next = * head ;
   * head = new node ;
}
void move hode (struct node** dest ref, struct node **
                                          Source ref)
{
  if (*source ref == NULL)
   return;
   struct node * new node = *source ref ;
  *source ref = (* source ref) -> next ;
    new node -> next = * dest ref ;
    *dest ref = new node;
  }
  int  main (void)
{
  int keys[] = {1, 2, 3} ;
  int n= size of (keys) / size of (keys[0]);
struct node *a = NULL;
for (int i= n-1 ; i>=0 ; i--)
     push (&a, keys[i]);
struct node *b= NULL;
for (int i=0 ; i<n; i++)
     push(&b, 2* keys[i]);
```

```
Move node (&a, &b);

Printf ("First list");
Print list (a);
Printf ("second list");
Printlist (b);
return 0;
}
```