Lakshmi. S. kumar
1BM19CS078

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
{
int info;
struct node *link;
};
typedef struct node * Node;
Node getnode();
{
Node x
x = (NODE) malloc (sizeof (struct node));
if (x == Null)
printf ("Memory full\n");
exit (0);
}
return x;
}
void freenode (NODE x
{
free(x);
}

NODE insert_rear (NODE first, int item)
{
NODE temp, cur;
temp = getnode();
temp->info = item
temp->link = Null
if (first == Null)
return temp;
cur = first
```

Scanned with CamScanner

```
while (cur -> link != Null)
  cur = cur -> link
  cur -> link = temp
    return first
}
NODE delete rear (NODE first
{
  NODE cur prev;
    if (first == Null)
    {
    printf ("List is empty cannot delete \n");
  return first;
}
if (first -> link == = Null)
{
  printf ("item deleted %d \n" first -> info)
  free (first)
    return Null;
    }
  prev = Null
    cur = first;
    while (cur -> link != Null)
    {
    prev = cur
    cur = cur -> link
    {
    printf ("Itemdeleted at rear end is %d", cur->info)
free (cur);
prev -> link = Null
  return first;
}
NODE insert_pos (int pos, NODE first)
```

```
{
NODE temp, cur, prev
 int count
temp = getnode
temp -> info = item
temp -> link = Null
 if (first = Null && pos = = 1)
{
return temp;
}
if (first = = Null)
{
printf ("Invalid position \n")
return first;
}
if (pos = = 1)
{
tem -> link = first
  first = temp
return temp;
}
count = 1
prev = Null
 cur = first
 while (cur != Null && count != pos)
{
prev = cur
 cur = cur -> link
  count ++
}
if (count = = pos)
{
```

```
prev -> link = temp
temp -> link = cur
}
printf ("Invalid position \n");
    return first;
    }
NODE delete_pos (int pos, NODE first)
{
NODE cur;
NODE prev;
int count, flag=0
if (first == NULL || pos<0
printf ("Invalid position \n");
    return NULL;
  {
printf ("Invalid position \n");
    return NULL;
    }
if (pos == 1)
{
cur = first
first = first -> link;
freenode (cur).
    return first;
    }
prev NULL
  cur = first
  count = 1
  while (cur != NULL
    {
  if (count == pos ) { flag = 1; break;
count ++
```

```
prev cur
 cur = cur -> link
  }
  if (flag == 0)
{
printf ("Invalid position \n");
return first;
}
 print ("Item deleted at given position is %d
-> info);
 prev -> link = cur -> link;
    free node (cur);
 return first;
 }
 void display (NODE first)
 {
 NODE temp;
 if (first == Null
 printf(" list is empty cannot display ite
   m \n")
for (temp = first temp != Null; temp -> link
{
printf ("%d ", temp -> info
  }
  }

int main()
  int item, choice pos
NODE first = Null
for (;;)
  NODE
  {
```

```
printf("\n Insert_rear\n 2. Delete_rear\n");
printf(3. Insert_info position\n 4. Delete_info
       5: Display_list\n 6: Exit\n");

printf("Enter the choice");
scanf("%d", &choice);
switch(choice)
{
case 1 printf("Enter the item at rear end.\n");
scanf("%d", &item);
first=insert_rear(first, item)
break;
case 2: first = delete_rear(first);
break;
case 3: printf("Enter the item to be inserted
at given position:\n");
scanf("%d", pos)
first = insert_pos(item, pos, first);
break;
case 4: printf("Enter the position\n");
scanf("%d", &pos);
first=delete_pos(pos, first)
break;
case 5 display(first);
break;
default: exit(0);
break;
}
}
return 0;
}
```

8 program

```
#include<stdio.h>
#include<stdlib.h>
{
int info
struct node *NODE;
NODE*
};
struct node * link;
};
typedef struct node *NODE;
NODE x
x(NODE) malloc size (struct node));
if (x==Null
{
print f ("mem full\n")
exit (0)
};
return x;
}
NODE insert_rear (NODE first, int item)
{
NODE temp cur
temp =getnode
temp > info = item;
temp link = Null;
if (first == = Null):
return temp;
cur = first
while (cur > link! = Null)
Cur = cur-> link
return first)
}
```

```
}
void display(NODE first)
{
NODE temp;
if (first == NULL)
printf("list is empty").
for (temp=first; temp = NULL; temp= temp->link
{
printf("%d", temp->info);
}
}
NODE concat (NODE first, NODE second
{
NODE cur;
if (first != NULL)
{
temp = first;
first = first -> line
temp -> link = cur;
cur = temp;
}
return cur;
}
int main ()
{
int item choice, pos, i, n;
NODE first = NULL, a, b;

for (;;)
{
printf("1 insert- front In 2. concat In 3.
reverse 4 display In 5 exit
```

```
Case 1 printf ("enter the item \n");
  Scanf ("%d", &item);
  first = insert_rear (first, item);
break;
Case 2 : printf ("enter the no of nodes in \n")
Scanf ("%d", &n);
a = Null
for (i=0 ; i<n; i++
{
printf ("enter the item \n")
  Scanf ("%d", &n);
  b = Null
  for (i=0 ; i<n, i++)
  {
Printf ("enter the nodes in 2 \n");
  Scanf ("%d", &n)
  b = Null
  full (i=0 , i<n, i++)
  }
printf ("enter the item \n")
Scanf ("%d, & item);
b = insert_rear (b, item);
}
a concate (a, b);
  display (a)
  break;
  Case 3 : first = reverse (first)
    display (first);
    break
    case 4 display (first);
    break;
    default exit (0);
```

}
}
}

Queue

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
int info;
struct node *link
};
typedef struct node *node
    node get node()
      {
    NODE x
x=(NODE) malloc (sizeof(struct node));
    if (x==NULL
{
printf("memefull\n");
exit(0);
}
return x
}
void free node (NODE x)
{
free (x);
}
NODE insert_rear
{
NODE temp, cur;
temp = get node()
temp -> info = item
temp -> link = NULL
if (first == NULL)
```

```
return temp
cur = first
while (cur -> link != NULL)
 cur = cur -> link
 cur -> link temp;
 return first;
}
NODE delete_front (NODE first)
{
NODE temp;
if(first == NULL)
 {
printf (list is empty they cannot
    delete \n")
return first;
}
temp = first
temp = temp -> link
printf ("item deleted at front-end is = %d\n
    first -> info);
return temp;
 }
void display (NODE first)
 {
  NODE temp;
  if (first == NULL)
  printf ("list empty cannot display
    item \n");
for (temp = first; temp != NULL; temp =
  link)
}
printf ("%d \n" temp -> info)
```

```
}
}
int main
int item, choice, pos;
NODE first = NULL
for(;;)
{
printf("1: Insert rear 2: Delete front 3:
    Display list 4 Exit \n");
Scanf("%d", &choice
Switch (choice
{
case 1: printf("enter the item at rear end\n");
Scanf("%d", &item);
Switch (choice)
{
case 1 printf("Enter the item at rear end\n");
Scanf("%d", &item);
first = insert_rear(first, item);
break;
Case 2: first = delete_front(first);
break;
default exit(0);
break;
}
}
}
```

Stack

```
#include <stdio.h>
//#include <conio.h>
//#include <alloc.h>
#include <stdlib.h>
  struct node
{
 int info
 struct node * link
  };
 type of struct node * NODE
  NODE getnode()
   NODE x;
  x=(NODE) malloc (sizeof(struct node));
  if (x=NULL)
 {
printf ("memory full \n");
  exit(0);
  }
  return x
 }
 void freenode(NODE x)
   {
  free(x);
 }
 NODE insert_front(NODE first, int item)
{
 NODE temp;
  temp =getnode;
 }
  NODE deletion_front(NODE first)
  {
   NODE temp;
```

```
if (first == Null)
{
printf ("Stack is empty cannot delete\n")
return first
}
temp = temp -> link
printf ("\n item delete at front end is %d"
        first -> info)
free (first
return temp;
}
void display (NODE first)
{
NODE temp;
if (first == Null)
printf ("Stack empty cannot display items\n")
for (temp = first, temp != Null; temp = temp ->
link)
{
printf ("%d", temp -> info)
}
}
int main ()
{
int item, choice, pos;
NODE first = Null
for (;;)
{
printf ("\n 1 Insert front\n 2 Delete front
        3 Display - list 4 Exit\n")
Printf ("Enter the choice\n")
Scanf ("%d" & choice
```

```
switch (choice)
  {
case 1 : printf (" enter the item at front end\n")
scanf ("%d" & item);
first = insert_front (first, item);
break;
case 2: first = delete_front (first);
 break.
 default: exit (0);
 break;
```