```
#include<stdio.h>
#include<alloc.h>
#include<process.h>
struct node
{ int info;
  struct node *link;
};
typedef struct node *NODE;
 NODE getnode()
{ NODE = x
```

Scanned with CamScanner

```
x = (NODE)malloc (sizeof (struct node));
if (x==NULL)
{ printf ("mem full\n");
  exit(0);
}
return x; //creates a node for me and return the
}                                          address*/
void freenode(NODE x)
{ free(x);
}
NODE insert_front(NODE first, int item)
{ NODE temp;
  temp = getnode(); /* obtain the node from available list*/
  temp->info = item; /*Insert the item in new node created*/
  temp->link = NULL;
  if (first == NULL) /* checking for my list is empty */
  return temp; /* if my list is empty the temp node
                  will be first nod */
  temp->link = first; /* attach the new node at front
                         end of list */
  first = temp;
  return first;
}
NODE delete_front (NODE first)
{ NODE temp;
  if (first == NULL)
{ printf("List is empty cannot delete\n");
  return first;
}
```

Scanned with CamScanner

```
temp = first;
temp = temp -> link;
printf("item deleted at front-end is =%d", first->info);
free(first);
return temp;
}
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;
cur -> link = temp;
return first;
}
NODE delete_rear (NODE first)
{
NODE cur, prev;
if (first == NULL)
{
printf("item deleted is %d\n", first -> info);
printf("list is empty can't delete\n");
return first;
}
```

Scanned with CamScanner

```
if (first -> link == NULL)
{printf ("item deleted is %d\n", first -> info);
 free(first);
 return NULL;
}
prev = NULL;
cur = first;
while (cur -> link != NULL)
{prev = cur;
 cur = cur -> link;
}
printf ("item deleted at rear - end is %d", cur -> info);
free (cur);
prev -> link = NULL;
return first;
}
```

Scanned with CamScanner

```
void display (NODE first)
{
  NODE temp;
  if (first == NULL)
  Pf("list empty can't display items\n");
  for(temp=first; temp!=NULL; temp=temp->link)
  {
    Pf ("%d", temp -> info);
  }
}
```

Scanned with CamScanner

```
int main()
{
  int c, item, pos;
  int n, i;
  int choice;
  NODE first = NULL, sec, fir;
  for(;;)
{
printf("1: Stack\n2: Queue\n3: Exit\n");
printf("Enter choice");
scanf("%d", &c);
switch(c)
{ case 1: printf("Stack\n");
    for(;;)
  {printf("1: Insert-rear\n2: Delete-rear\n3: Display
                                    \n4: Exit\n");
    scanf("%d", &choice);
    switch(choice)
{
  case 1: printf("Enter item at rear-end");
          scanf("%d", &item);
          first = insert_rear(first, item);
          break;
  case 2: first = delete_rear(first);
          break;
  case 3: display(first);
          break;
  default: exit(0);
}}
  break;
```

Scanned with CamScanner

```
case 2: printf("Queue\n");
   for(;;)
{ printf("1:Insert_rear\n2:Delete_front\n3:Display
                                    \n4:Exit\n");
   printf("Enter choice");
   scanf("%d", &choice);
   switch(choice)
{ case 1: printf("Enter the item at rear-end");
          scanf("%d", &item);
          first = insert_rear(first, item);
          break;
   case 2: first = delete_front(first);
          break;
    case 3: display(first);
          break;
    default: exit(0);
          break;
    }
  }
   break;
 case 3: exit(0);
   default: printf("Invalid choice\n");
 } }
}
```

Scanned with CamScanner