linked list

```
#include <stdio.h>
#include <stdlib.h>
struct node{
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_front(NODE first, int item ){
NODE temp;
temp = getnode();
temp -> info = item;
temp -> link = NULL;
if (first == NULL)
return temp;
temp -> link = first;
first = temp;
return first;
}
Node delete_front(NODE first){
NODE temp;
```

```
if (first == Null) {
printf ("list is empty connot delete \n");
return first;
}
temp = first;
temp = temp -> link;
printf ("item deleted at front and is %d \n",
        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 ("list is empty connot delete \n");
 return first;
}
 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;
}
NODE insert_pos (int item, 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) {
temp -> link = first;
first = temp;
return temp;
}
```

```
count = 1;
prev = Null;
cur = first;
while (cur != Null && count != Pos) {
per = cur;
cur = cur -> link
count ++;
}
 if (count == Pos) {
prev -> link = temp;
temp -> link = cur;
return first;
}
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;
}
 if (Pos == 1) {
 cur = first;
first = first -> link;
 freenode (cur);
return first;
}
prev = Null
cur = ~~cur -> link~~ 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;
}
printf ("item deleted at a given Position is %d",
        cur -> info);
prev -> link = cur -> link;
free node (cur);
return first;
}
void display (node first) {
NODE temp;
if (first == NULL)
printf ("list empty cannot display items \n");
for (temp = first; temp != NULL; temp = temp -> link) {
 printf ("%d \n", temp -> info);
}
}
void main ()
{
int item, choice, key, pos;
int count = 0;
```

```
NODE first = NULL;
for (;;) {
printf ("\n1: Insert rear\n2: Delete rear\n3:
insert front\n4: delete front\n5: insert info
postion\n6: delete info postion\n7: display\n8
. 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 ("\nEnter the item at front end\n");
scanf ("%d, &item);
first = insert_front (first, item);
break;
case 4: first = delete_front (first);
break;
Case 5: printf (" Enter the item to be inserted at
a given Position\n");
Scanf ("%d", &item);
printf (" Enter the Postion\n");
Scanf ("%d", &Pos);
first = insert_Pos (item, pos, first);
break;
Case 6: Printf (" Enter the Postion\n");
Scanf ("%d", &Pos);
first = delete_Pos (pos, first);
break;
```

```
case 7 : display(ptr);
break;
default : exit(0);
break;
}
}
}
```