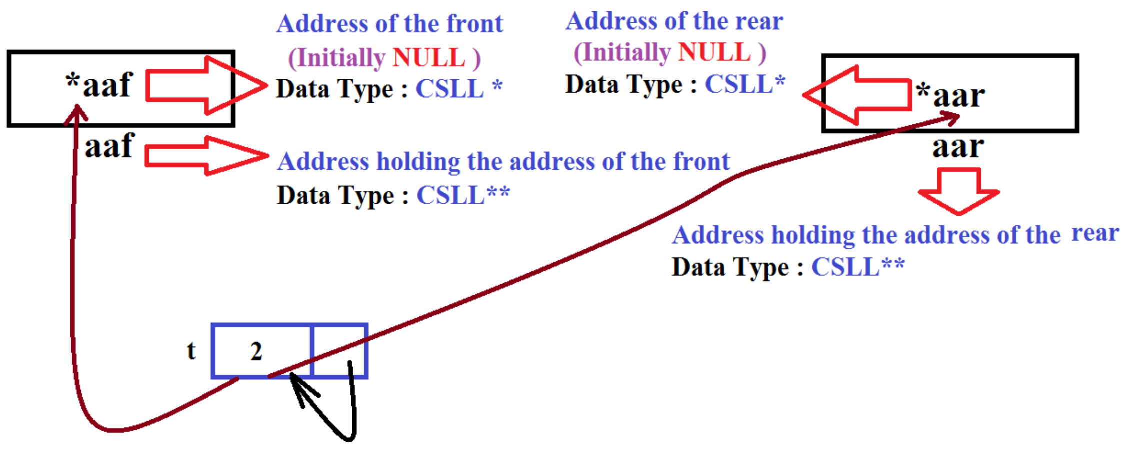
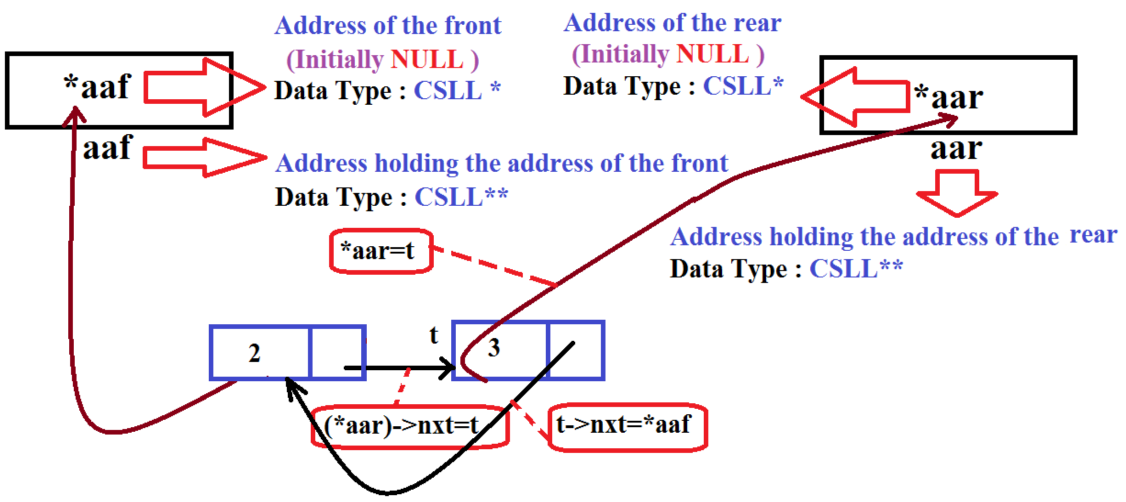
Circular Queue using Circular Linked List

Dr. Rahul Das Gupta

```
#include<stdio.h>
#include<stdlib.h>
#define EMPTY_ERROR_CODE -9999

typedef struct scll
{
  int data;
  struct scll *nxt;
}CSLL;

void initialisation(CSLL **, CSLL **);
int is_empty_queue(CSLL *);
void insert_in_sorted_order(CSLL **, CSLL **, int);
void insert_at_the_rear(CSLL **, CSLL **, int);
int deletion_from_the_front(CSLL **, CSLL **);
void deletion_specified_item(CSLL **, CSLL **, int);
void display(CSLL *);


void initialisation(CSLL **aaf, CSLL **aar)
{
  *aaf=NULL;
  *aar=NULL;
}

int is_empty_queue(CSLL *af)
{
   return(af = = NULL);
}

void insert_in_sorted_order(CSLL **aaf, CSLL **aar, int n)
{
  CSLL *cur,*prv,*head=NULL,*t;
  t=(CSLL *)malloc(sizeof(CSLL));
t->data=n;
if(*aaf==NULL)
{
 *aaf=t;
 *aar=t;
 t->nxt=t;
 return;
}

for(cur=*aaf,prv=NULL; cur!=head && n>cur->data; cur=cur->nxt)
{
   head=*aaf;
   prv=cur;
}
t->nxt=cur;

if(prv)
   prv->nxt=t;
else
 {
    (*aar)->nxt=t;
    *aaf=t;
 }
}
```

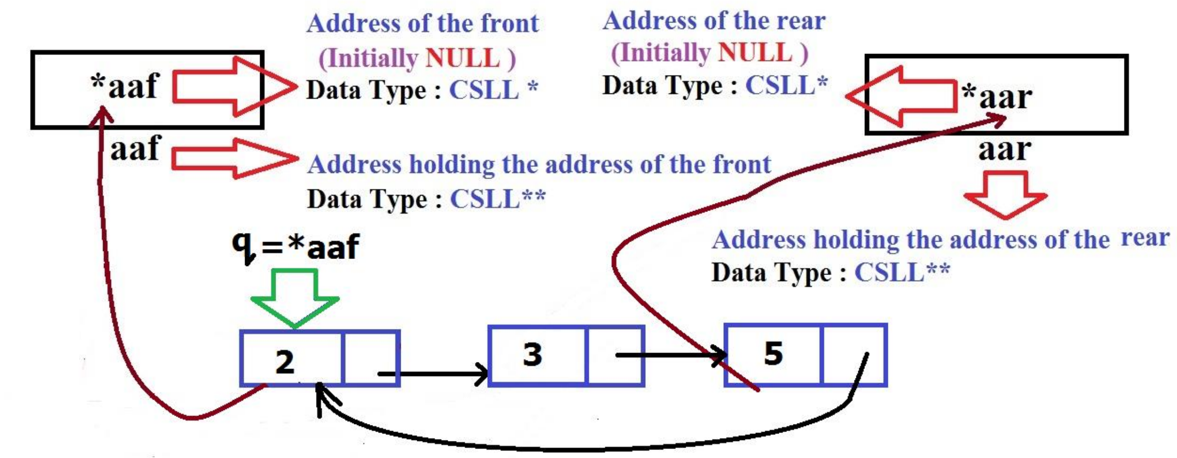


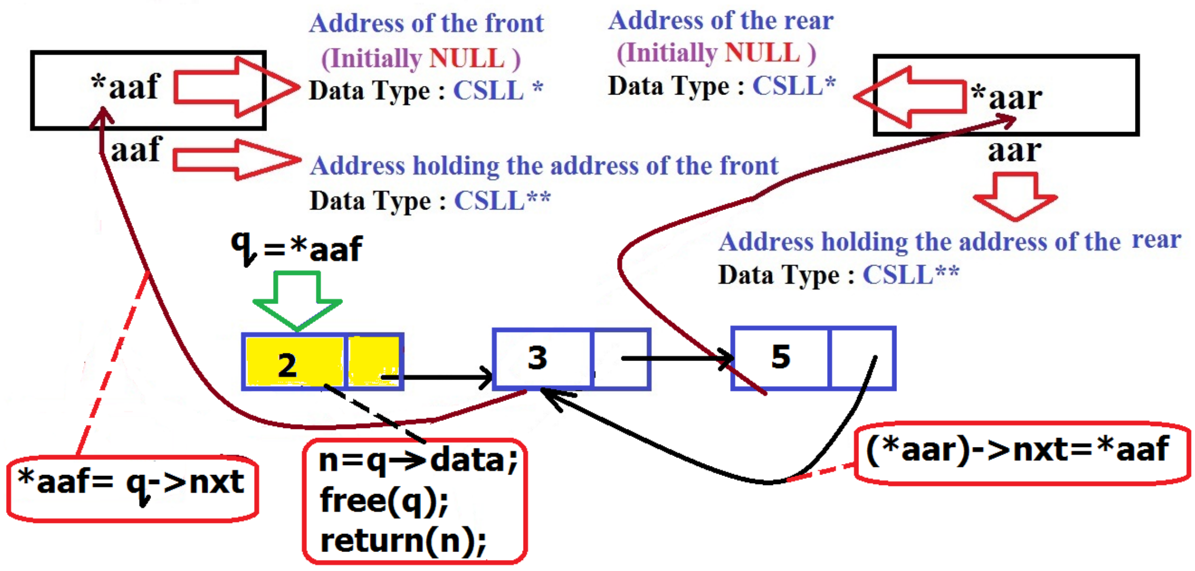
Case1: **When the first node is inserted in the Circular Queue.**



Case 2: When any other node is inserted after the first node in the Circular Queue.

```
void insert_at_the_rear (CSLL **aaf, CSLL **aar, int n)
{
  CSLL *cur,*prv,*head=NULL,*t;
  t=(CSLL *)malloc(sizeof(CSLL));
  t->data=n;
  if(*aaf==NULL)
  {
   *aaf=t;
   *aar=t;
   t->nxt=t;
   return;
  }
  else
   {
      (*aar)->nxt=t;
      t->nxt=*aaf;
      *aar=t;
   }
}
```





```
int deletion_from_the_front(CSLL **aaf, CSLL **aar)
{
  CSLL *q;
  int n;
  if(*aaf==NULL)
  {
     printf("\n Empty Queue...");
     return EMPTY_ERROR_CODE;
  }
  q=*aaf;
   n=q->data;
   if(*aaf == *aar) /*When there is a single node in the Circular
Queue. */
   {
      *aar = NULL;
      *aaf = NULL;
    }
 else
{
  *aaf=q->nxt;
  (*aar)->nxt=*aaf;
 }
  free(q);
  return n;
}

void deletion_specified_item(CSLL **aaf, CSLL **aar, int n)
{
 CSLL *cur, *prev, *head=NULL;
 int found=0;

 if(*aaf==NULL)
 {
    printf("\n Empty list...");
    return;
 }

 for(cur=*aaf, prev=NULL; cur!=head; cur=cur->nxt)
 {
  head=*aaf;
  if(n==cur->data)
  {
      found=1;
      if(cur == *aar)
             *aar=prev;
      /* If the node to be deleted is rear, then its previous node will
be declared as the new rear.*/
      if(prev)  /*Case 1: When prev exists.*/
               prev->nxt=cur->nxt;
      else if (cur->nxt != cur) /*Case 2: When prev does not exist and
the Circular Queue contains more than one node.*/
         {
         *aaf=cur->nxt;
          if (*aar)
               (*aar)->nxt=*aaf;
      /* The next part of the rear node will point to the front node.*/
          }
       else  /*Case 3: There is no prev and only a single node is
present in the Circular Queue. */
        {
           *aaf=NULL;
           *aar=NULL;
         }
      free(cur);
      printf("\n Successfully deleted the item %d...",n);
      return;
  }
  else
      prev=cur;
 }
 if(found==0)
    printf("\n Item not found...");
}

void display(CSLL *af)
{
 CSLL *cur,*head=NULL;
 if(af==NULL)
 {
   printf("\n Empty list...");
   return;
 }
 printf("\n");
 for(cur=af; cur!=head ; cur=cur->nxt)
  {
   head=af;
   printf("\t %d", cur->data);
  }
 printf("\n");
}

void main( )
{
 CSLL *f=NULL, *r=NULL;
 int k;
 initialisation(&f, &r);
 insert_in_serial_order(&f, &r, 10);
 insert_in_serial_order(&f, &r, 2);
 insert_in_serial_order(&f, &r, 5);
 insert_in_serial_order(&f, &r, 12);
 insert_in_serial_order(&f, &r, 8);
 insert_in_serial_order(&f, &r, 4);
 insert_in_serial_order(&f, &r, 6);
 k=deletion(&f, &r);
 k=deletion(&f, &r);
 deletion_specified_item(&f,&r,4);
 display(f);
}
```