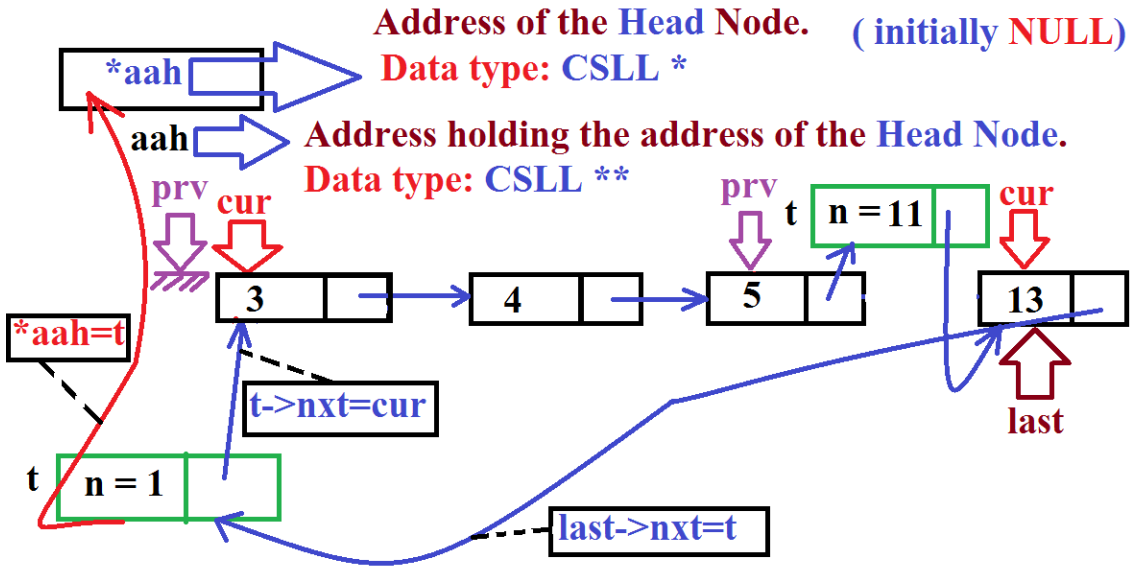
Circular Single Pointer Linked List

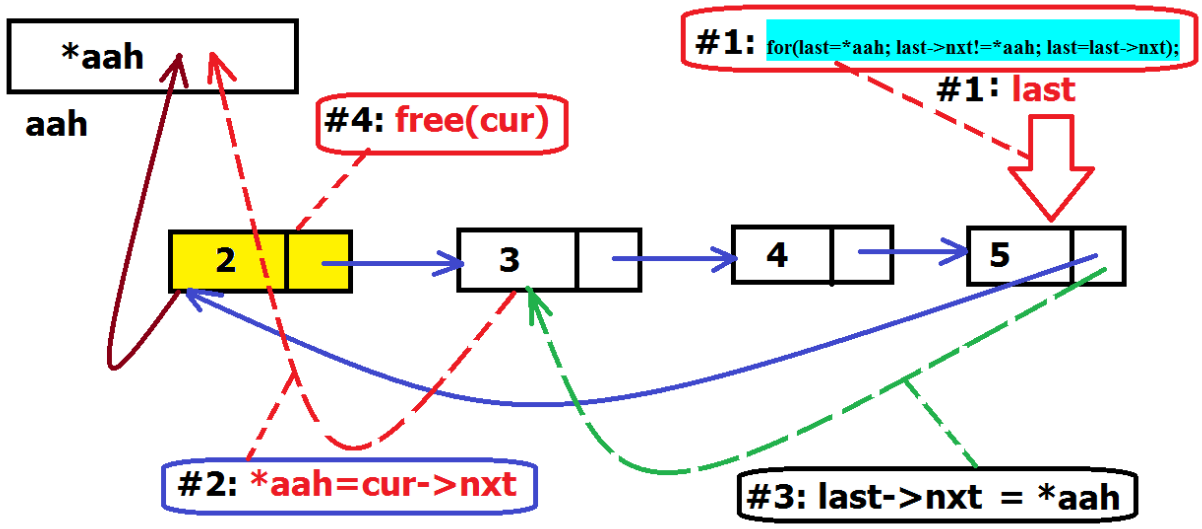
Dr. Rahul Das Gupta

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

                /* Definition of Self Referential Structure*/
typedef struct scll
{
  int data;
  struct scll *nxt;
}CSLL;
                        /*Prototype Deceleration*/
void initialisation(CSLL **);
void insert_in_sorted_order(CSLL **, int);
void insert_in_serial_order(CSLL **, int);
void deletion(CSLL **, int);
void display(CSLL *);

                        /*Function Definitions*/
void initialisation(CSLL **aah)
{
  *aah=NULL;
}

void insert_in_sorted_order(CSLL **aah, int n)
{
    CSLL *cur,*prv,*head=NULL,*last, *t;
    /* Creation of a new node t. */
    t=(CSLL *) malloc(sizeof(CSLL));
    t->data=n;
    if(*aah==NULL) /* In case of the first node.*/
    {
      *aah=t;
```

```
        t->nxt=t;
        return;
    }
/* Determination of the appropriate place of the newly created node
t . The newly created node t must be inserted between prv and cur.*/

/* A circular trip is required.  Start from the head node and finish at
head node.*/
  for(cur=*aah, prv=NULL; cur!=head && n>cur->data; cur=cur->nxt)
  {
         head=*aah; /* Set the pointer to the head node every time. */
         prv=cur; /* Before visiting the next node, the current node must
                                                be marked as the previous node. */
  }
/* The newly created node t must be inserted between prv and cur.*/
  t->nxt=cur;

  if(prv) /* Check whether previous node exists or not.*/
     prv->nxt=t;
 else /* There is no previous node. The newly created node t will be
added at the beginning.*/
   {
     /* Identify the last node that is linked with the head node.*/

        for(last=*aah; last->nxt!=*aah; last=last->nxt);

        last->nxt=t; /* The last node must be linked to the new node.*/

        *aah=t; /*The new node t must be declared as the head node.*/
   }
}
```




```
void insert_in_serial_order(CSLL **aah, int n)
{
  CSLL *cur,*t;
  t=(CSLL *)malloc(sizeof(CSLL));
  t->data=n;
  if(*aah==NULL)
  {
      *aah=t;
      t->nxt=t;
      return;
  }

  for(cur=*aah;cur->nxt!=*aah; cur=cur->nxt); /*Go to the last node.*/
  cur->nxt=t; /* Link the last node to the newly created node.*/
  t->nxt=*aah; /* Link the newly created node to the head node.*/

}
```




```
void deletion(CSLL **aah, int n)
{
 CSLL *cur, *prev, *head=NULL, *last;
 int found=0;
 if(*aah==NULL)
 {
    printf("\n Empty list...");
    return;
 }
/* A head node to head node circular trip is required.  */
 for(cur=*aah, prev=NULL; cur!=head; cur=cur->nxt)
 {
  head=*aah;
  if (n= =cur->data) /*Element to be deleted found at the location cur. */
  {
     found=1; /*Set the value of indicator found as 1.*/

     if (prev) /*Check whether the previous node exists or not.*/
       prev->nxt=cur->nxt; /* Removing the link between prv->nxt
                                  and cur   */
      /*In order to remove the current node, previous node must be
linked to the node immediately next to current.*/
```

```
    else if (cur->nxt! =* aah)
    {
      /* Identify the last node that is linked with the head node.*/
      for(last=*aah; last->nxt!=*aah; last=last->nxt);
      /*There is no previous node. Node to be deleted is head node.*
      *aah=cur->nxt;
      /* The node next to current node will be marked as head node.*/
      last->nxt=*aah;
      /* The next part of the last node will point to the head node.*/
     }
    else
        *aah=NULL;
     free(cur); /*Deallocation of memory at the location cur.*/
     printf("\n Successfully deleted the item %d...",n);
     return;
  }
  else
      prev=cur;
 }
 if (found = = 0)
        printf("\n Item not found...");
}



void display(CSLL *ah)
{
   CSLL *cur,*head=NULL;
   printf("\n");

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

```
/* A head node to head node circular trip is required.  */

  for(cur=ah; cur!=head ; cur=cur->nxt)
  {
        head=ah;
        printf("\t %d", cur->data);
  }
  printf("\n");
}

void  main( )
{
    CSLL *l=NULL;
    initialisation(&l);
    insert_in_sorted_order(&l,10);
    insert_in_sorted_order(&l,2);
    insert_in_sorted_order(&l,5);
    insert_in_sorted_order(&l,12);
    insert_in_sorted_order(&l,8);
    insert_in_sorted_order(&l,4);
    insert_in_sorted_order(&l,6);
    deletion(&l,8);
    display(l);
}
```