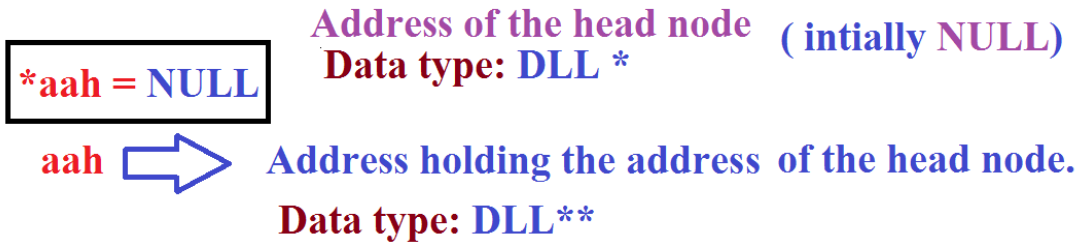
Linear Linked List With Double Pointer

Dr. Rahul Das Gupta

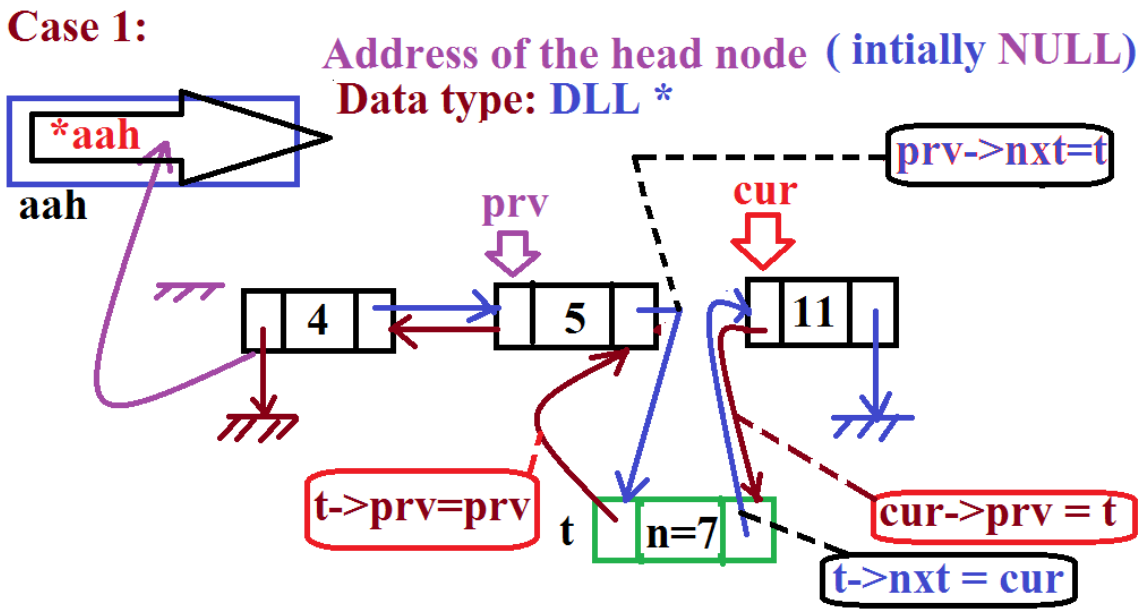
```
#include<stdio.h>
#include<stdlib.h>

/* Definition of Self Referential Structure */

typedef struct dll
{
  int data;
  struct dll *prv;
  struct dll *nxt;
}DLL;

/* Prototype Declarations */

void initialisation(DLL **);
void insert_in_sorted_order(DLL **, int);
void insert_in_serial_order(DLL **, int);
void deletion(DLL **, int);
void display(DLL *);
void reverse_display(DLL *);

/* ==========================================
aah = Address holding the address of the head node
(Datatype of aah: DLL **)
   *aah = Address of the head node
(Datatype of *aah: DLL *)
==========================================*/
```




```
void initialisation ( DLL **aah)
{
  *aah=NULL;
}
```

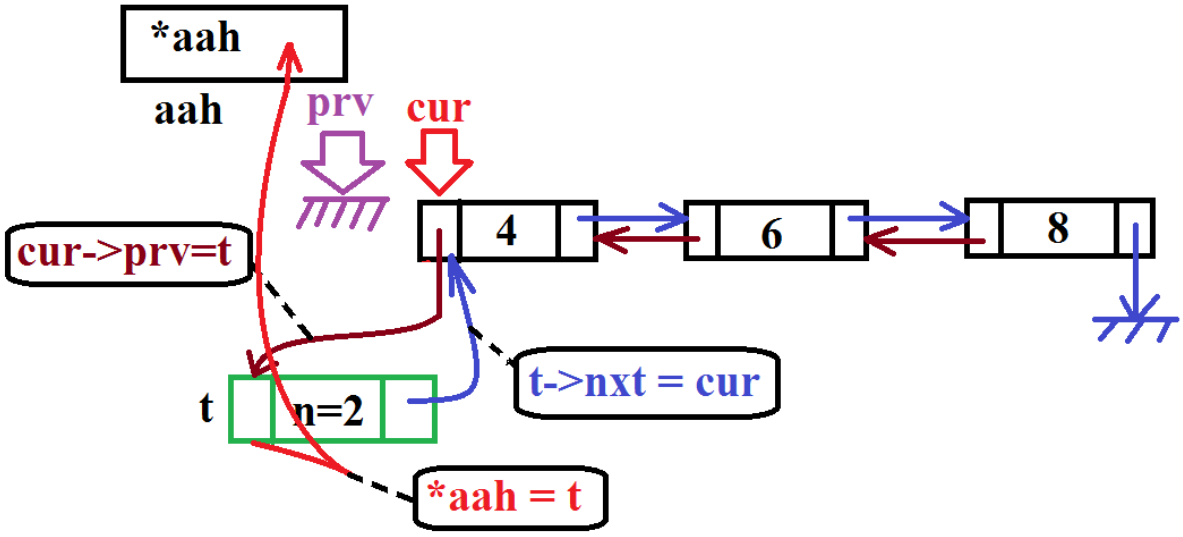
```
void insert_in_sorted_order(DLL **aah, int n)
{
  DLL *cur,*prev, *t;
  /* Allocating memory space for temporary node */
  t=(DLL *)malloc(sizeof(DLL));
  t->data=n;
  t->prv=NULL;
  t->nxt=NULL;

/*Detection of the appropriate node previous to the node containing
the value n.*/
  for(cur=*aah, prev=NULL; cur && n>cur->data; cur=cur->nxt)
      prev=cur;

  t->nxt=cur; /* cur containing the value immediately next to n.*/
  if(cur) /* check whether cur exist or not. */
        cur->prv=t;    /* this statement will lead to Segmentation Fault if
                       the pointer cur is non-existing that is cur==NULL*/

  if(prev) /* check whether prev exist or not. */
  {
```

```
        prev->nxt=t; /* this statement will lead to Segmentation Fault if
                        the pointer prev is non-existing that is prev == NULL*/
        t->prv=prev;
  }
  else /* when value of  n is less than all the existing elements of  the list. */
     *aah=t; /* inserted as the first element */
}
```



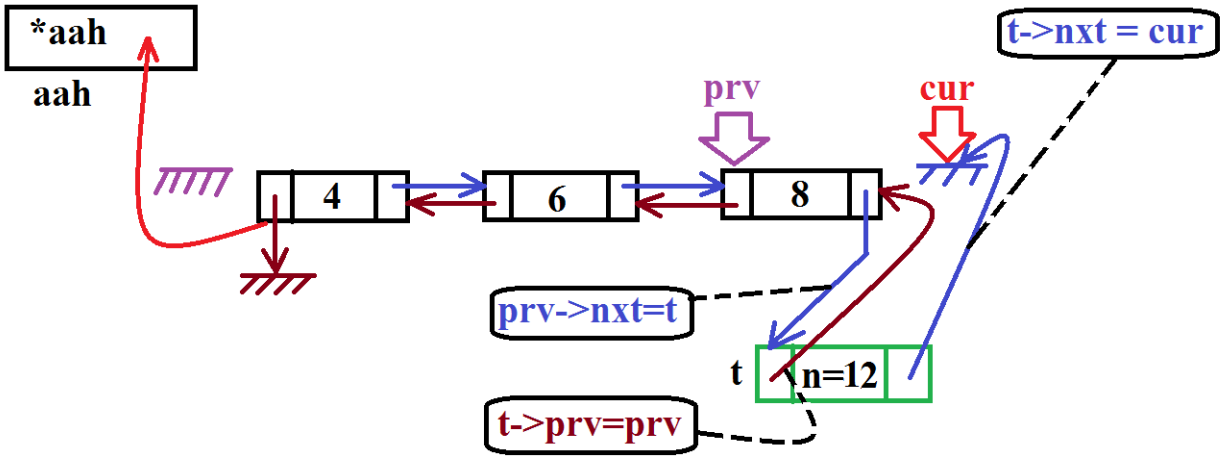
/* Case 1: The item to be inserted within the list has both the previous and next nodes. */

Case 2:



/* Case 2: The item to be inserted within the list has no previous node. Hence, item will be added at the beginning of the list.*/

Case 3:



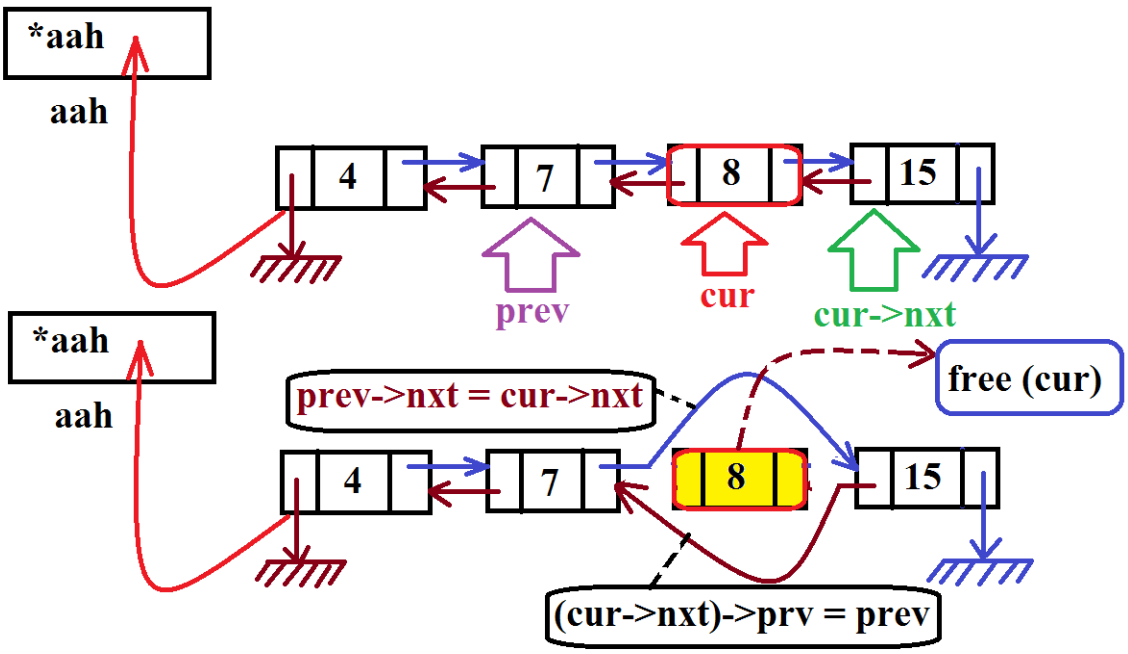
/* Case 3: The item to be inserted within the list has no next node. Hence, item will be added at the end of the list.*/

/* This function is almost same as insert_in_sorted_order exept the condition inside the for loop */

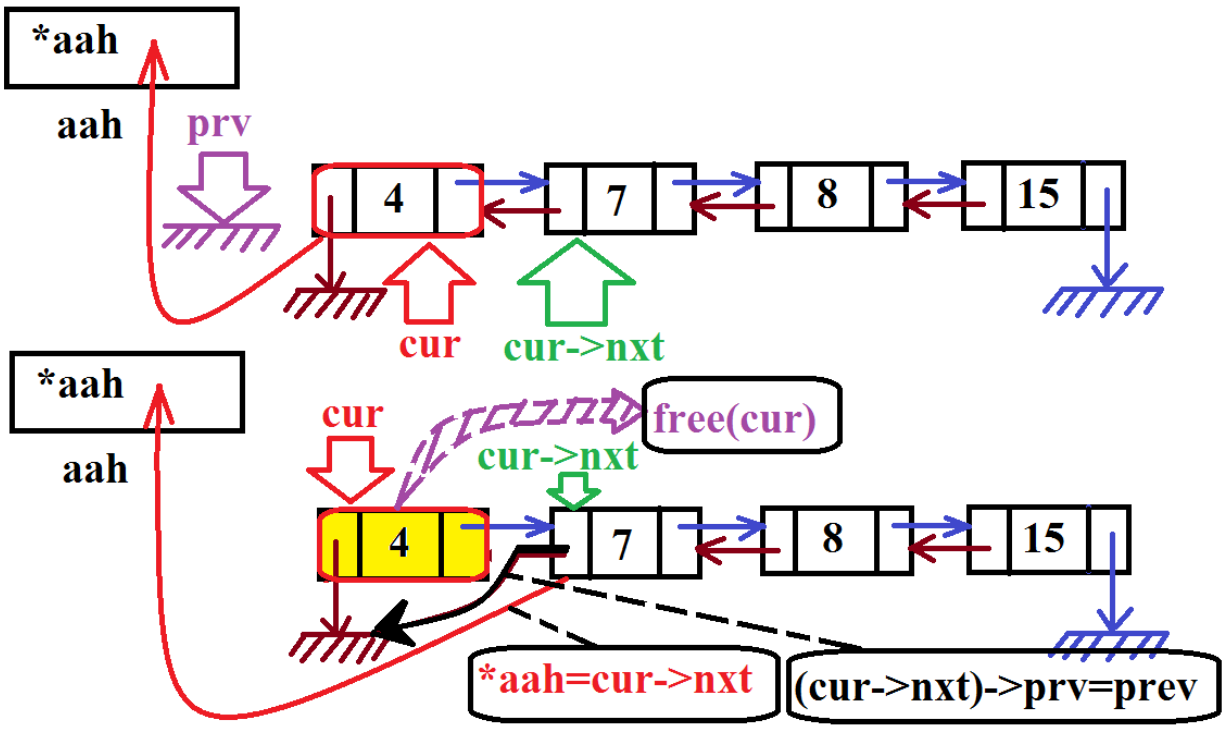
```
void insert_in_serial_order(DLL **aah, int n)
{
  DLL *cur,*prev, *t;

  t=(DLL *)malloc(sizeof(DLL));
  t->data=n;
  t->prv=NULL;
  t->nxt=NULL;

  for(cur=*aah, prev=NULL; cur ; cur=cur->nxt)
      prev=cur;

  t->nxt=cur;
  if(cur)
     cur->prv=t;

  if(prev)
  {
     prev->nxt=t;
     t->prv=prev;
  }
  else
     *aah=t;
}
```

```
void deletion(DLL **aah, int n)
{
    DLL *cur,*prev;
    int found=0; /* initialise the indicator found as 0. */
    if(*aah==NULL)
    {
       printf("\n Empty list...");
       return; /* Nothing to do. Just return.*/
     }
    for(cur=*aah, prev=NULL; cur; cur=cur->nxt)
    {
       if(n = = cur->data) /*element to be deleted found in the list.*/
       {
           found=1; /* set the indicator found as 1. */
           if(prev) /* check whether prev exist or not. */
           {
                prev->nxt=cur->nxt; /*prev->next was cur before execution of this
                  statement, remove the connection between prev->next and cur */
                       /* this statement will lead to Segmentation Fault if
                                                          prev == NULL*/

              if(cur->nxt) /* check whether cur->nxt exist or not. */
                     (cur->nxt)->prv=prev;
                       /* this statement will lead to Segmentation Fault if
                                                       cur->nxt == NULL*/
            }
           else
           {
              *aah=cur->nxt; /* cur is removed.*/
               if(cur->nxt) /* check whether cur->nxt exist or not. */
                     (cur->nxt)->prv=NULL;
                       /* this statement will lead to Segmentation Fault if
                                                       cur->nxt == NULL*/

           }
           free(cur); /* Deallocate the memory at the location cur. */
           printf("\n Successfully deleted the item %d...",n);
           return;
```

```
        }
        else /* go to the next element in the list. */

            prev=cur; /* mark the present node cur as the prev. */
    }
    if (found==0)
        printf("\n Item not found...");
}
```



/*Case 1: The item to be deleted is an internal node within the list.*/



/*Case 2: The item to be deleted is the first node in the list.*/

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

```
void reverse_display(DLL *ah)
{
 DLL *cur;
 printf("\n");
 if(ah==NULL)
 {
    printf("\n Empty list...");
    return;
 }
 for(cur=ah;cur->nxt;cur=cur->nxt); /* Go to the end node of the list .*/
/*It is a 'Do Nothing Loop'.*/
 for(; cur; cur=cur->prv)
    printf("\t %d", cur->data);
  printf("\n");
}



void main()
{
 DLL *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);
 reverse_display(l);
}
```