Double Pointer Linked List

Rahul Das Gupta

```
typedef struct d_linked_list
{
 int data;
 struct d_linked_list *prv,*nxt;
}NODE;


void insert_in_sorted_order (NODE ** aah, int n)
{
  NODE *cur, *prv,*t;
  t=(NODE *) malloc(sizeof(NODE));
  t->data = n;
  t->prv = NULL;
  t->nxt = NULL;
  for( cur=*aah, prv=NULL; cur && n>cur->data; cur=cur->nxt)
                prv=cur;
  t->nxt=cur;
  if(cur !=NULL)  /* Make sure that the pointer cur actually exists.*/
            cur->prv=t;
  if (prv) /* Make sure that the pointer prv actually exists.*/
  {
            prv->nxt=t;
            t->prv=prv;
   }
  else  /* insert at the beginning. */
  {
            *aah=t;
             t->prv=NULL;
  }
}


/*      Sequential Insertion                        */

/*  Remove the condition n>cur->data from the function
insert_in_sorted_order */
```

```
void insert_at_end(NODE ** aah, int n)
{
  NODE *cur, *prv,*t;
  t=(NODE *) malloc(sizeof(NODE));
  t->data = n;
  t->prv = NULL;
  t->nxt = NULL;
  for( cur=*aah, prv=NULL; cur ; cur=cur->nxt)
                prv=cur;
  t->nxt=cur;
  if(cur !=NULL)  /* Make sure that the pointer cur actually exists.*/
            cur->prv=t;
  if (prv) /* Make sure that the pointer prv actually exists.*/
  {
            prv->nxt=t;
            t->prv=prv;
   }
  else  /* insert at the beginning. */
  {
            *aah=t;
             t->prv=NULL;
  }

}

void insert_at_beginning(NODE **aah, int n)
{
  NODE * t;
  t=(NODE *) malloc(sizeof(NODE));
  t->data = n;
  t->prv = NULL;
  t->nxt = *aah;
  if(*aah != NULL)
            (*aah)->prv=t;
  *aah=t;
}
```

```
void deletion (NODE **aah, int n)
{
   int  found=0;
   NODE *prv,*cur,*t;
   for(prv=NULL,cur=*aah; cur; )
   {
      if (n==cur->data)
      {
          found=1;
          t=cur;
          if(prv==NULL)
           {
             *aah=cur->nxt;
             if(cur->nxt !=NULL)
                    (cur->nxt)->prv=NULL;

          }
          else
          {
             prv->nxt=cur->nxt;
             if(cur->nxt !=NULL)
                    (cur->nxt)->prv=prv;
          }
         free(t);
         printf("\n Successfully deleted %d",n);
      }
      else
      {
        prv=cur;
        cur=cur->nxt;
      }
    }
    if(!found)
    {
          printf("\n  Item not found…");
          exit(1);
    }
}
```

```
void forward_display (NODE *ah)
{
    NODE * cur;
    for (cur=ah; cur; cur=cur->nxt)
            printf("\t %d",cur->data);

}

void reverse_display (NODE *ah)
{
    NODE * cur;
    for (cur=ah; cur->nxt; cur=cur->nxt); /* go to the last node */
    for (;cur; cur=cur->prv)
         printf("\t %d",cur->data);
}
```