**[finding intersection point in two linked lists program](http://chanduthedev.blogspot.in/2012/07/finding-intersection-point-in-two.html)**

Here is C code for finding merge point in two linked lists using different methods.  [Click here](http://chanduthedev.blogspot.in/2012/07/linked-list-intersection.html) for explanation.

[?](http://chanduthedev.blogspot.in/2012/07/finding-intersection-point-in-two.html)

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  struct node  {   int info;   struct node \*next;  };    typedef struct node Node;    Node\* append(Node \*h, int info)  {   Node \*tempHead=h;   Node \*newNode = (Node\*)malloc(sizeof(Node));   newNode->info = info;   newNode->next = NULL;   if(tempHead == NULL) // if the list has zero elements. make new node as a head   {    h=newNode;   }   else if(tempHead->next==NULL)  // if the list is having only one node   {    tempHead->next = newNode;   }   else   {    Node \*tempNode = h;    while(tempNode->next != NULL)  // if the list has more than one node, so moving to the last node    {     tempNode = tempNode->next;    }    tempNode->next = newNode; // appending the new node at the end   }   return h;  }      /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  for displaying the nodes in the list  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void display(Node \*h)  {   Node \*temp = h;   while (temp->next!=NULL)   {    printf("%d->",temp->info);    temp = temp->next;   }   printf("%d\n",temp->info);  }        /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   this function returns the no. of nodes in the linked list  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int get\_count(Node \*h)  {   int count=0;   if(h == NULL)    return count;   while(h!=NULL)   {    count++;    h=h->next;   }   return count;    }    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  This method is more efficient and simple to implement. Get the count of nodes  in L1 and L2. take the difference as 'diff' and traverse 'diff' nodes in the  bigger list and then traverse parallelly both L1 and L2 and check each node  in L1 n L2. If matches that is the merge point.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  Node\* get\_merge\_node(Node \*h1, Node\* h2, int diff)  {   int i;   for(i=0;i<diff;i++)   {    h1 = h1->next;   }   while(h1!=NULL || h2!=NULL)   {    if(h1 == h2)     return h1;    h1=h1->next;    h2=h2->next;   }   return NULL;  }    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   This is basic method. using two loops we can compare each node in the L1 with each node in L2  It will take O(n \*m ) where n is length of L1 and m is length of L2  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  Node\* merge\_point\_basic\_method(Node\* h1, Node\* h2)  {      while(h1)      {          Node \*tmp = h2;          while(tmp)          {              if(h1 == tmp)                  return h1;              tmp=tmp->next;          }          h1=h1->next;      }      return NULL;  }    main()  {   Node \*head1 = NULL;   Node \*head2 = NULL;   Node \*merge\_node = NULL;   int i,count1,count2,merge\_info,method;   for (i=1;i<=10;i++)   {    head1 = append(head1,i\*10);   }   for (i=1;i<=5;i++)   {    head2 = append(head2,i\*2);   }   Node \*temp1 = head1;   for (i=1;i<=3;i++)   {    temp1 = temp1->next;   }   printf("List one elements ....\n");   display(head1);   printf("List two elements ....\n");   display(head2);     Node \*temp2 = head2;   while(temp2->next!=NULL)    temp2=temp2->next;   //making merge point for two linked lists, comment below line for removing merge point   temp2->next = temp1; //making merge point for two linked lists   printf("List two elements after making merge point ....\n");   display(head2);   printf("Enter the method in which you need to find the merge point\n");   printf("1. Basic method\n2. Using difference of nodes\n3. Exit\n");   scanf("%d",&method);   switch(method)   {    case 1:      merge\_node = merge\_point\_basic\_method(head1, head2);      break;    case 2:      count1 = get\_count(head1);      printf("list one length is %d\n",count1);      count2 = get\_count(head2);      printf("list two length is %d\n",count2);      if(count1 > count2)      {       //head1 list is bigger so passing head1 first       merge\_node = get\_merge\_node(head1,head2, count1-count2);      }      else      {       //head2 list is bigger so passing head2 first       merge\_node = get\_merge\_node(head2,head1, count2-count1);      }      break;    default: exit(0);   }     if(merge\_node)    printf("merge point data is %d\n",merge\_node->info);   else    printf("two lists are different\n");  } |