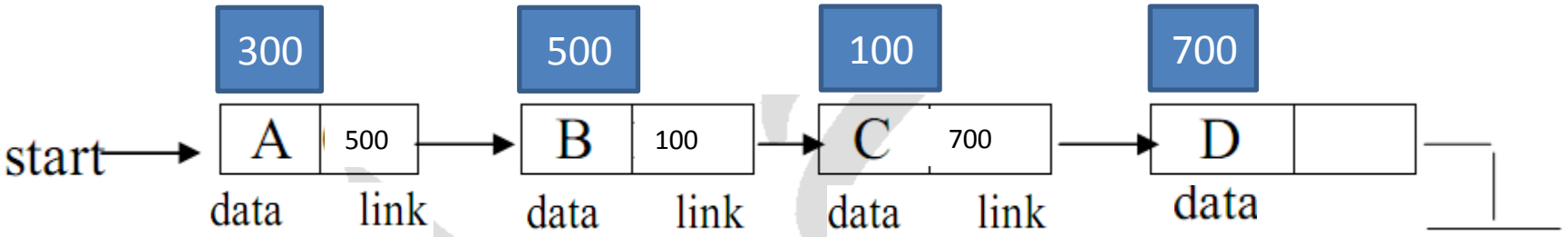
القائمة الموصولة linked list

هي مجموعة من العناصر (العقد) التي كل منها يحتوي البيانات data والمؤشر link الذي يشير الى العنصر (العقدة) التالي في القائمة.
فالعنصر x يتكون من الجزأين link,data

X:

Data(x)	Link(x)

مثال: لنأخذ قائمة موصولة



محتويات حقل المؤشر للعنصر الثاني هو(100) ويدل على موقع العنصر الثالث:(link(B)=100

محتويات حقل المؤشر للعنصر الرابع هو(NULL)اي لا يوجد عنصر بعد العنصر الرابع

القوائم الخطية linked list

- القائمة الخطية هي نوع من انواع هياكل البيانات الخطية تستخدم لتنظيم البيانات في الذاكرة و كذلك في تمثيل / محاكاة انواع اخري من هياكل البيانات .

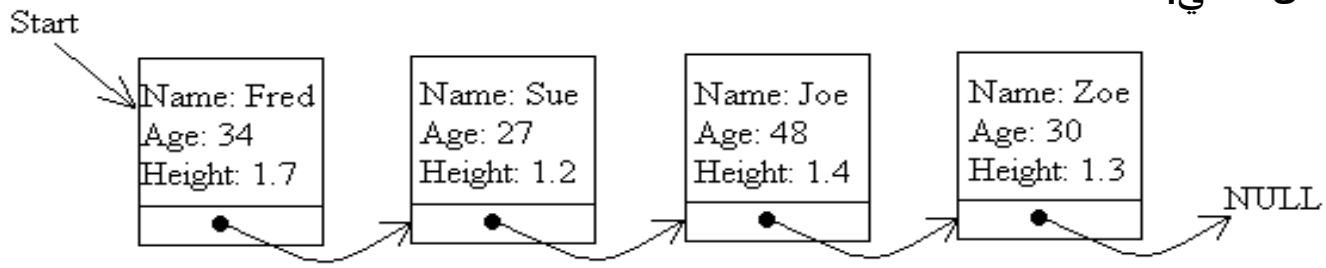
- ليس شرط ان تكون عناصر القائمة متجاورة في الذاكرة .

- تتكون من عدد من العناصر تسمي بالعقد **nodes** مرتبطة مع بعضها عن طريق مؤشرات.

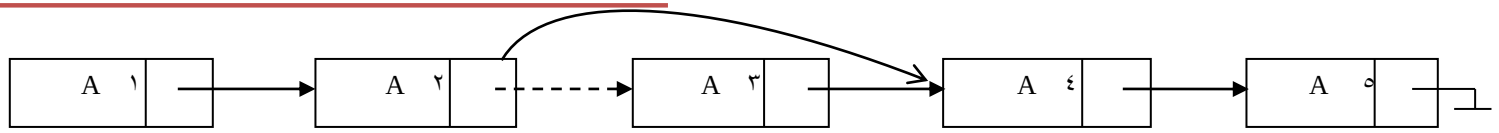
- العقدة (**node**) عبارة عن تركيبة struct أو كائن من طبقة تتكون من حقلين او اكثر من البيانات تسمي بـ 'data /Info' field و حقل آخر للمؤشر يعرف بـ 'Link/address' field .

- الحقل الاول يحمل قيم البيانات و الحقل الثاني يحمل عنوان العقدة التالية أو القيمة **NULL** اذا كانت القائمة فارغة أو نهاية القائمة.

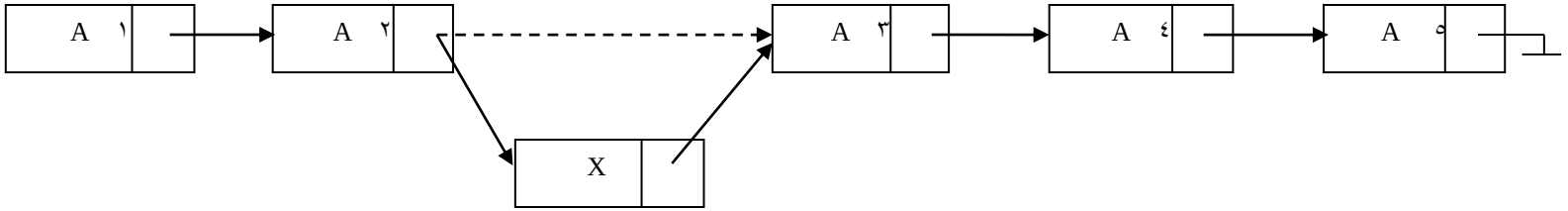
- تأخذ الشكل التالي:



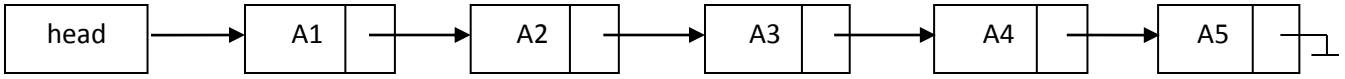
Deletion from a linked list



Insertion to a linked list



Linked List with a header



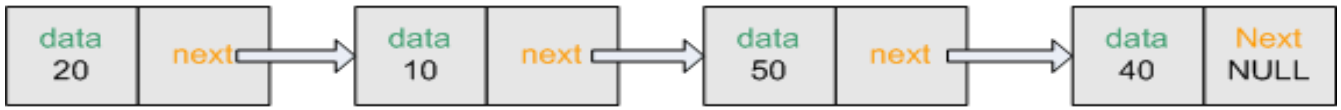
حيث head هو مؤشر للعقدة الأولي في القائمة.

مثال:

الاعلان عن عقدة (حلقة node) تحتوي علي البيان data + مؤشر للعقدة التالية

```
struct node
{
   int data;
   node *next;
};
```

The items of the list عناصر القائمة



Linked list

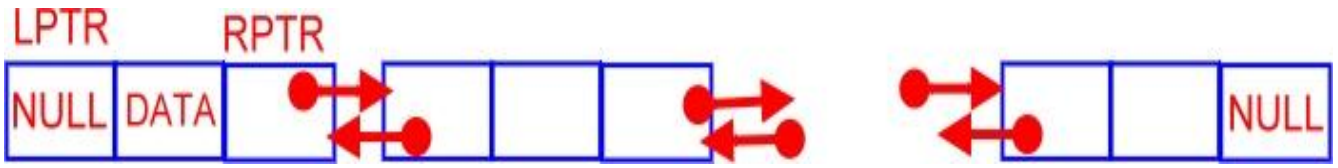
الاعلان عن عقدة Node تحتوي علي بيانات شخص (الاسم ، العمر ، الوزن)
+ مؤشر للعقدة التالية

```
struct Node
{
     char name[20];
     int age;
     float height;
     Node *next;              // Pointer to next node
};
Node *start_ptr = NULL;       // Start Pointer (root)
```

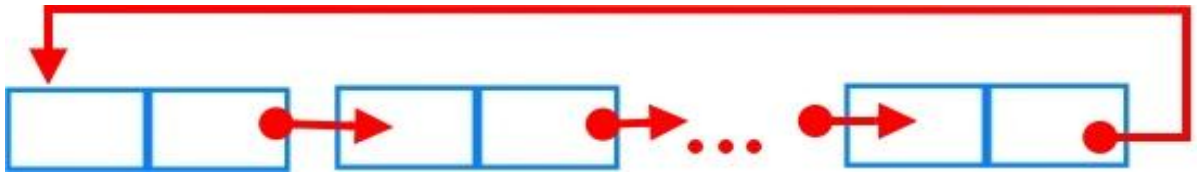
List Type's انواع القوائم

- Singly-Linked Lists
- Doubly-Linked Lists or Two way Linked List
- Circularly-Linked Lists
- Circularly-Doubly Linked Lists

قائمة خطية مزدوجة : Doubly-Linked List



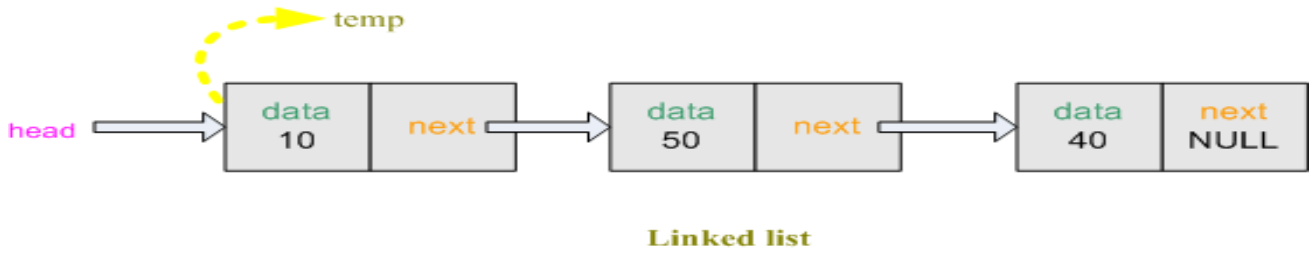
قائمة خطية دائرية Circularly-Linked List



عبارة typedef

```
Typedef  int  x;
   x  y;
   y=10;
   cout<<y<<endl;
```

Traverse/Display the list عبور القائمة

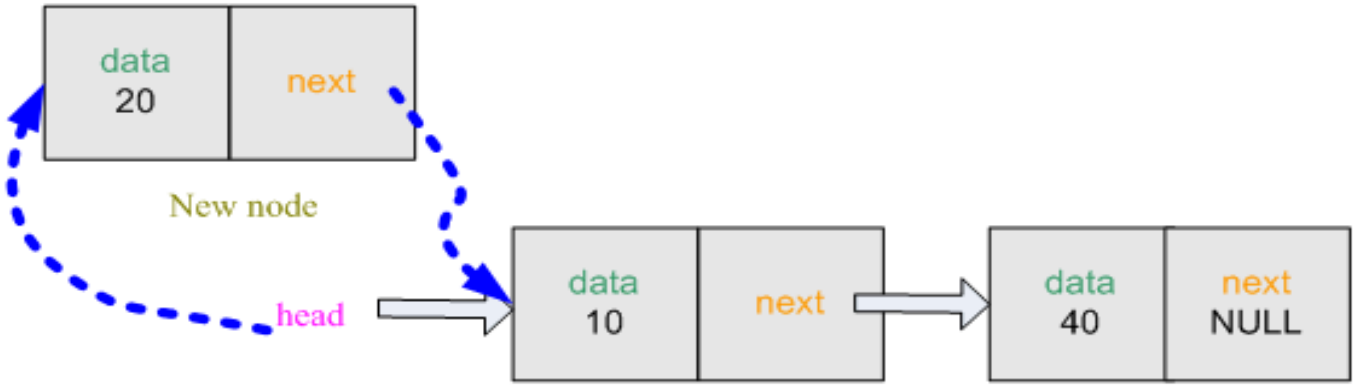


Linked list

1. Set a temporary pointer to point to the same thing as the start pointer.
2. If the pointer points to NULL, display the message "End of list" and stop.
3. Otherwise, display the details of the node pointed to by the start pointer.
4. Make the temporary pointer point to the same thing as the next pointer of the node it is currently indicating.
5. Jump back to step 2.

```
void display()
{
   node *temp=head;
   if(temp!=NULL)
   {
      while( temp!=NULL )
      {
       cout<< temp->data<<endl;
       temp = temp->next;
      }
  }
  else
     cout<<“\n empty  list”;
}
```

اضافة عقدة لبداية القائمة (Insert from front) :



Linked list

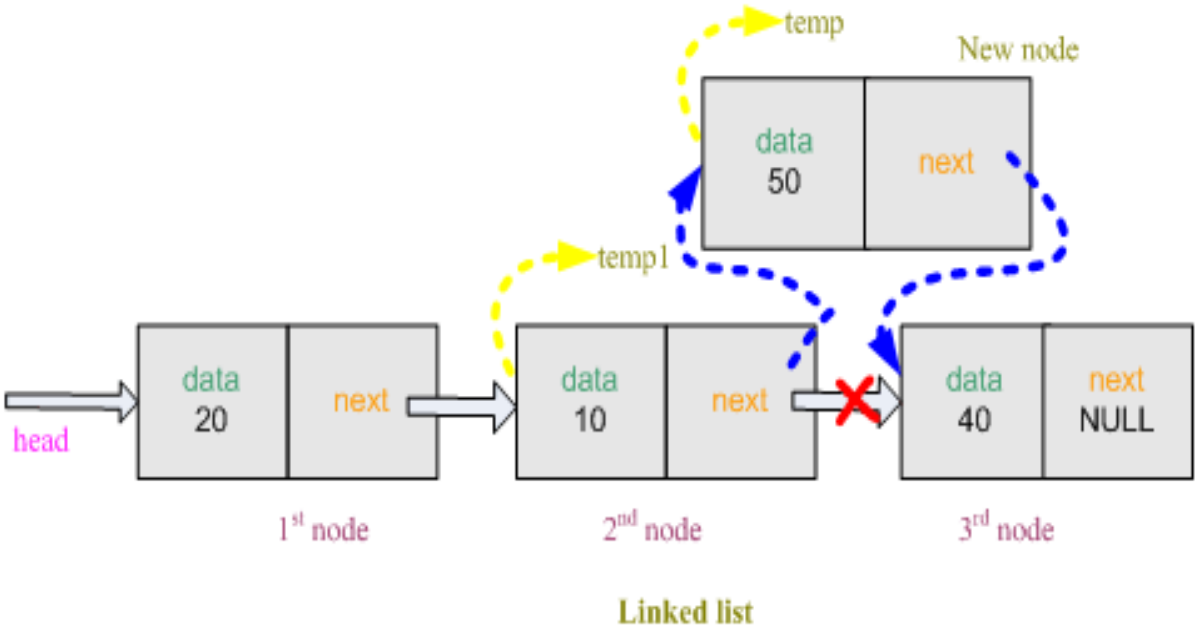
```
struct node
{
    int data;
    node *next;
};
struct node *head=NULL;        // LIST IS EMPTY
```

```
void insert_to_front ( )
{
   node *temp;
   if (head==NULL)
   {
      head=new node;
      cin>>head->data;
      head->next=NULL;
   }
   else
   {
      temp = new node;
      cin>>temp->data;
      temp->next=head;
      head = temp;
   }
}
```

اضافة عقدة لبداية القائمة (Insert from front) :

```
void main()
{
   insert_to_front( );
   insert_to_front( );
   display( );
}
```

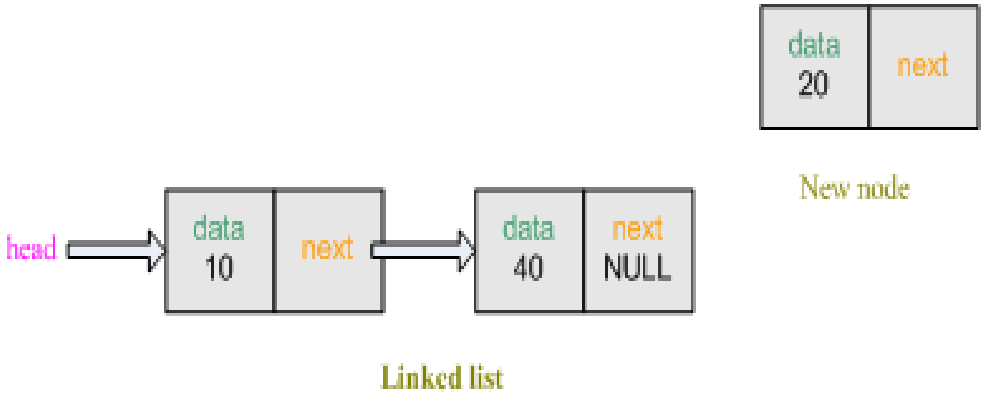
اضافة عقدة في أي مكان بالقائمة (بعد عقدة معينه) :

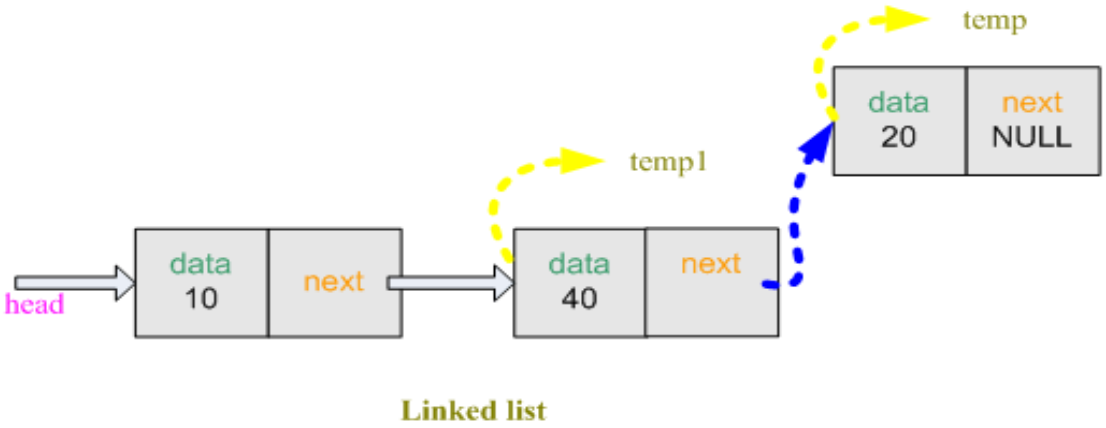


```
void insertAfter(int node_no )
{
   node *temp1;
   temp1 = head;
   for( int i = 1 ; i < node_no ; i++ )
   {
       temp1 = temp1->next;
       if( temp1 == NULL )
       {
            cout<<node_no<<" node is not exist"<< endl;
            break;
       }
   }
   node *temp;
   temp = (node*)malloc(sizeof(node));
   cin>>temp->data;
   temp->next = temp1->next;
   temp1->next = temp;
}
```

```
void main()
{
   insert_to_front( );
   insert_to_front( );
   traverse( );
   insertAfter(1);
   traverse( );
}
```

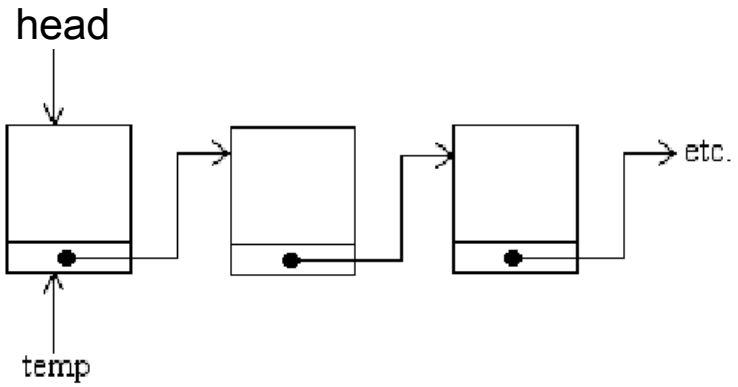
اضافة عقدة جديدة في نهاية القائمة Insert to back



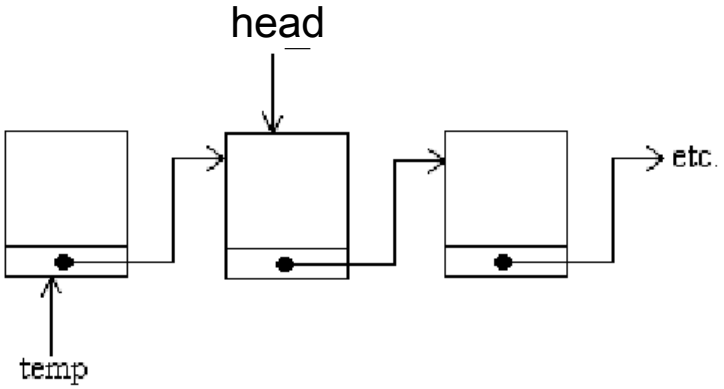


Deleting a node from the list

Here is the function that deletes a node from the start:



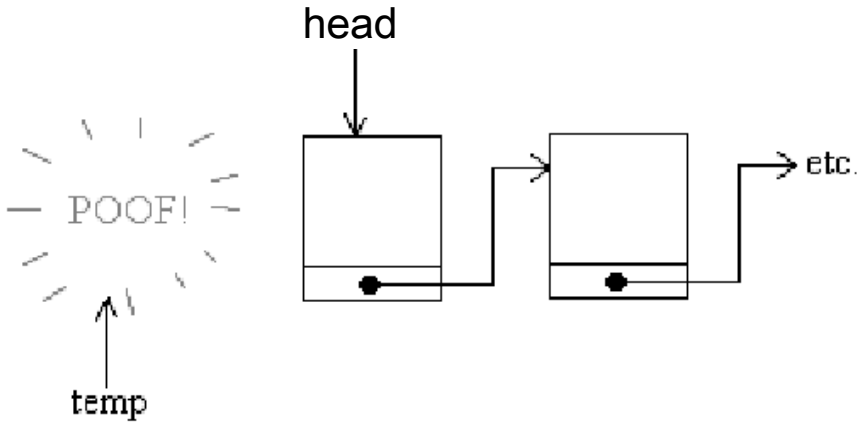

```
node *temp;
temp=head;
```




```
head = head->next;
```

أو

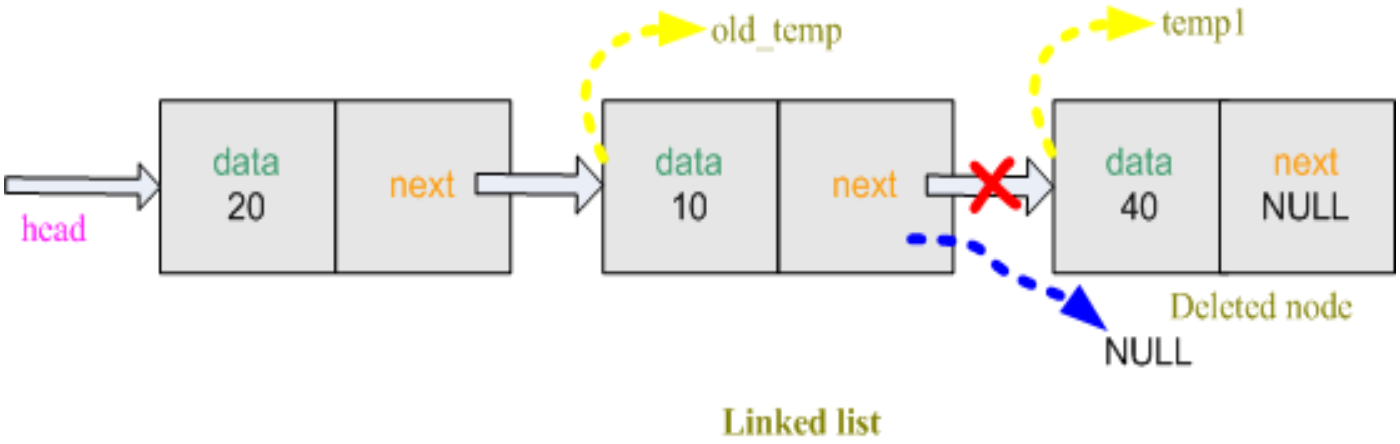
```
head = temp->next;
```




```
delete temp;     // Wipe out original start node
```

```
void delete_front( )
{
    node *temp;
    temp = head;
    head = temp->next;
    delete temp;
}
```

حذف عقدة من نهاية القائمة delete_from_back



Linked list

1. Look at the start pointer. If it is `NULL`, then the list is empty, so print out a "No nodes to delete" message.
2. Make `temp1` point to whatever the start pointer is pointing to.
3. If the `next` pointer of what `temp1` indicates is `NULL`, then we've found the last node of the list, so jump to step 7.
4. Make another pointer, `temp2`, point to the current node in the list (where `temp1` is pointing to).
5. Make `temp1` point to the next item in the list.
6. Go to step 3.
7. If you get this far, then the temporary pointer, `temp1`, should point to the last item in the list and the other temporary pointer, `temp2`, should point to the last-but-one item.
8. Delete the node pointed to by `temp1`.
9. Mark the `next` pointer of the node pointed to by `temp2` as `NULL` - it is the new last node.

```
void delete_from_back( )
{
      if ( head == NULL )   cout<<“empty list”;
      else
      {
            node *temp1 = head;
            if(temp1->next!=NULL)
            {
                  node *temp2;
                  while(temp1->next!=NULL)
                  {  temp2= temp1;          temp1 = temp1-> next;  }
                  temp2-> next = NULL;
                  delete  temp1;
            }
            else { delete  temp1;  head=NULL; }
      }
}
```