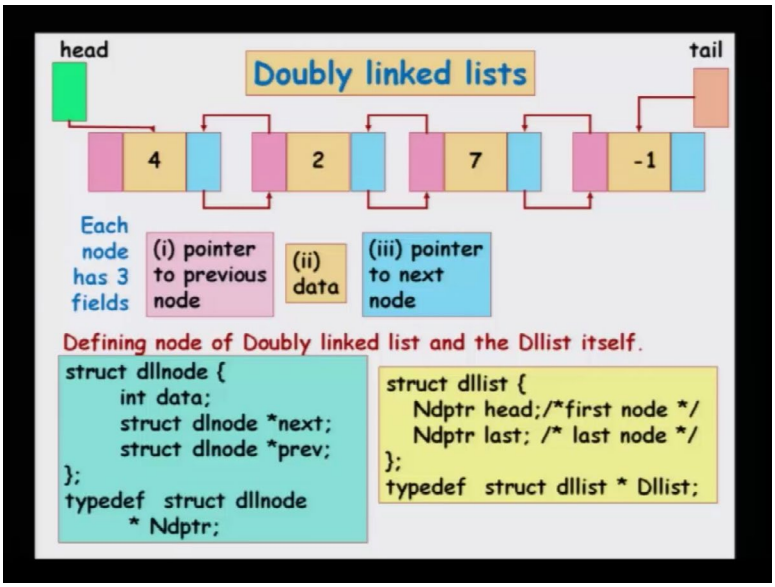
Introduction to Programming in C
Department of Computer Science and Engineering

In this lecture, we will see slightly more advanced data type, then a singly link list. We will briefly go over one or two functions to manipulate the data structure. The principle of manipulating the data structure for the other operations is similar.

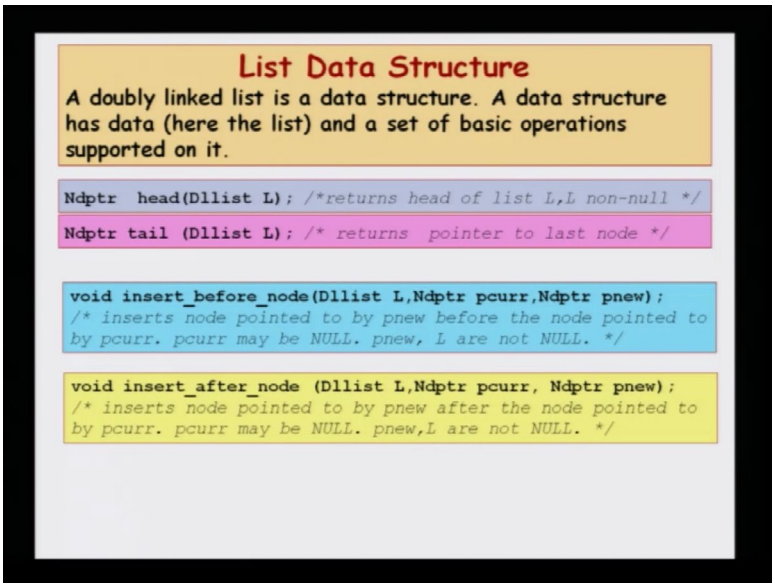
(Refer Slide Time: 00:18)



So, in the case of a singly link list, we have seen that every node has one link to its next neighbour, and we have seen this problem in a singly link list that if you are at a current node in a link list, you can always go forward, but there is no way to go back. There is one the only way took get to its previous node is to start all over again from the beginning of the list and traverse until you reach a list traverse, but traverse until you reach the previous node. So, we can easily remedy this by thinking of a data structure, a slightly more involve data structure where every node has two links; one... So, look at this node 2. So, it has two links; one is two it is neighbour successive neighbour. So, it is it is next node, there is another link which goes back to it is previous neighbour. So, in this data structure there are 2 links per node, therefore it is known as a doubly link list. And this list, obviously you can go from a current node, you can go forward or backward, so easily…

So, now the variation is this in each node has three fields; one is a pointer to the previous node, the second is the data in the node, and third is the pointer to the next node. So, how will the definition look like, it will say something like struct dllnode, doubly linked list node, it will have one filed which is data in data let us say, and then two nodes - struct dllnode next, and struct dllnode previous. So, one to the go to next node and another to go to the previous node. Now we will need two pointers typically for a doubly link list. One is the pointer to the beginning of list which is usually called the head, and then another to the end of list which is usually called tail. So, I will use a type def in order to the short term the name, I will just say typedef struct dllnode * Ndptr. And then I will say that the list has two node pointers; Ndptr heads, and node pointers last. So, doubly link list. Each node in the doubly link list has two list; one to its previous node, and another to is next node. And the list itself has two pointers; one to the beginning of the list - call the head, and another to the end of the list - call the tail.

(Refer Slide Time: 03:00)

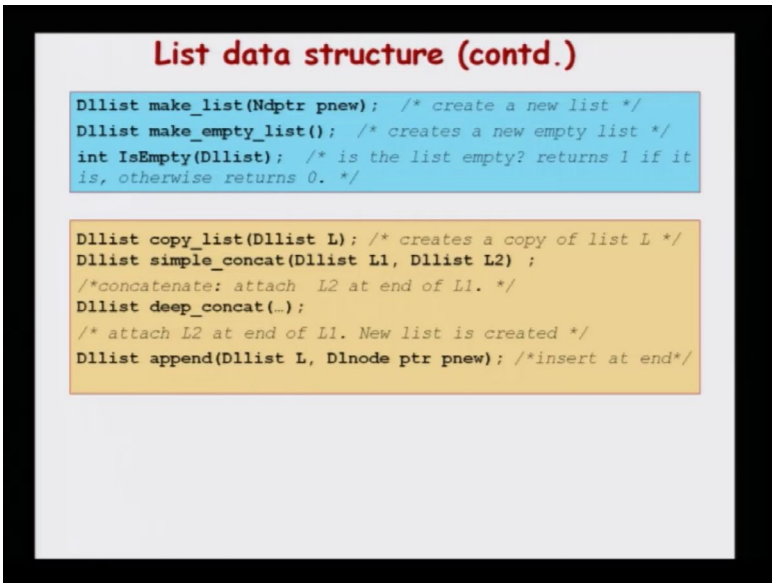


So, now a doubly link list is another data structure notice that we have seen two or three data structure so far arrays are one, which see already provides. We have already seen singly link list, now we have seen a third link list - third data structure which is a doubly link list. Now data structure has data and a bunch of operation defined on it. So, let us look at a typical operations that can be defined on a doubly link list, and we will go over the implementation of two or three them. So, Ndptr head. So, this is a function that should return the head of the list. Similarly node point of tail, they should return the tail

of the list. Insert before, so this is like a insert before node in the case of a singly link list. So, here we are given a current node and we have to insert before a current node in the doubly link list.

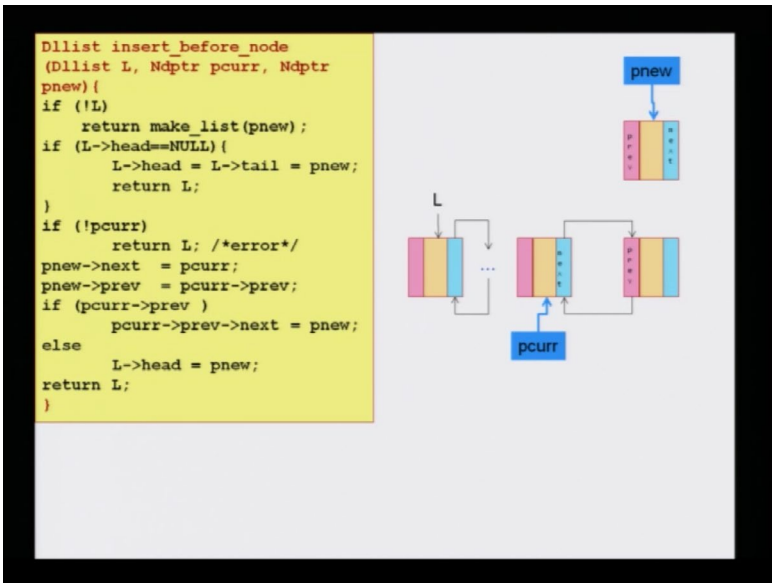
Notice thus this was difficult in singly link list, because there was no way to go from a current node to a previous node. We could always go to the next node. So, if I say that here is node and insert before that node in a singly link list is a difficult. You need some extra information, but in a doubly link list you have the current node and you can use the previous link in order to go before that. Insert_after_node also can be done this could also be done in a singly link list.

(Refer Slide Time: 04:41)



And then you can think of several other common like, you can think of a make node, you can think of a make_list with us single with a single node pointer two by pnew. You can make an empty list, you can check whether I given list is empty, you can a write functions to copy a doubly link list to an new doubly link list, you can concatenate to doubly link list. You can do a deep concat, we will see this in a features slide, you can append to link list, and so on. Similarly we can have insert, since we have insert functions we can also have delete functions, you can delete a particular node, you can extract a node in the sense that. So, delete would take out a node and free the memory allocated to the node, extract would just take out the node from the link list, but you retain the node, you can deleted entire list, and so on.

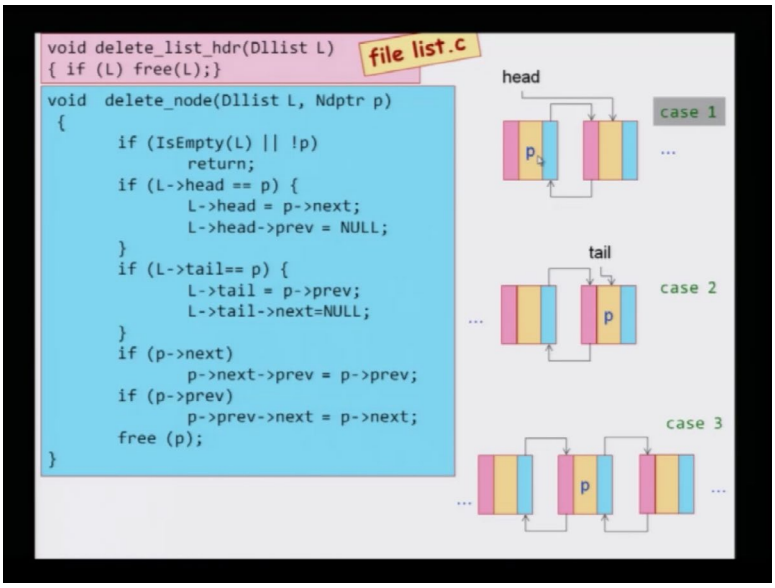
(Refer Slide Time: 05:52)



So, let us look at a couple of these functions; other functions can be return in similar manner. So, suppose let us take insert before load. This was a function that was not easy with the singly link list. So, I am given a link list L, and given a current node pcurr, and a new node to insert before the current node. So, what are the things to check? If the list is empty then insert before the current node just means that you create a new node, and return the new list. Now, if the head of the list is null, then you just say that now the new list contains only one load, L head will point new, L tail will point a new. So, if the list itself was null, then what you do is you create a new node, now the new list contains only one elements. So, the head will point to that and the tail will also point you that, and you return that. Now you come to the non trivial case, suppose there is a list; and the list has some elements. So, if pcurr is not equal to null then what to you do is, sorry if pcurr equal to null then you return L, this is an error. If pcurr is not equal to null then what you do is the following.

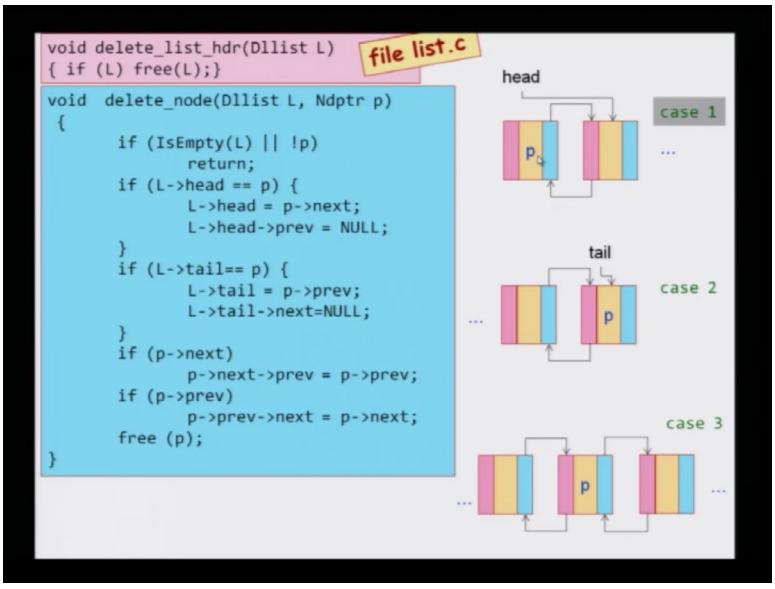
So, now you have to insert pnew in to the list. So, how do you do this? So, we say that the new nodes next will be... So, we are trying to insert pnew before pcurr. So, the new nodes next will. So, the new nodes next will be pcurr; pcurr previous will go to pnew. And so the pnew next will go to pcurr, and pcurr previous will go to pnew. Similarly we have to say that the previous node, the node before pcurr it has to point to pnew. So, pcurr previous that nodes next will good point to pnew, and then you return the new list, so this can be done by looking at pointers and handling pointers carefully.

(Refer Slide Time: 08:23)



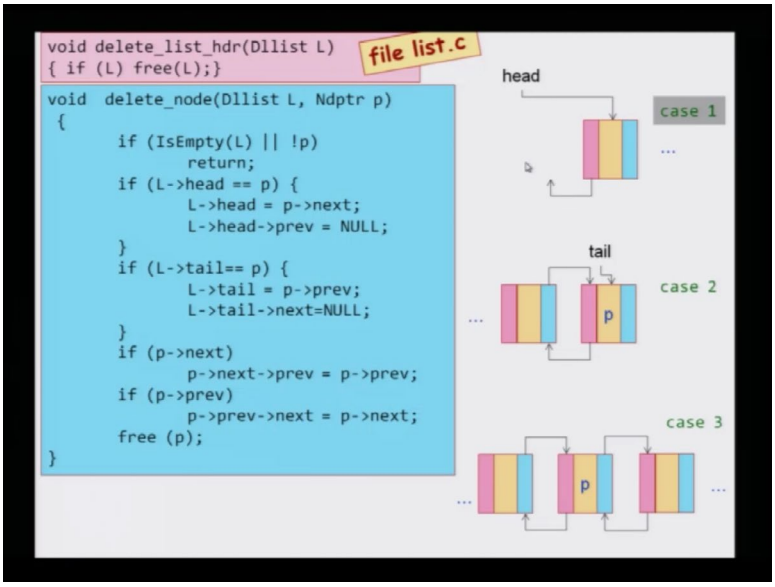
So, now let us see how to delete a particular node in a list. So, if you have to delete header of the list, then if there is a list you just delete the header, and you just free the entire list. Now if you have to delete a particular node in the middle of a list, what do you do? So, let us look at the various cases. So, in case one the node that you want to delete is the head of the list. So, in this case suppose you want to delete p, what would you do? You would make head point to the next element and free(p).

(Refer Slide Time: 09:10)



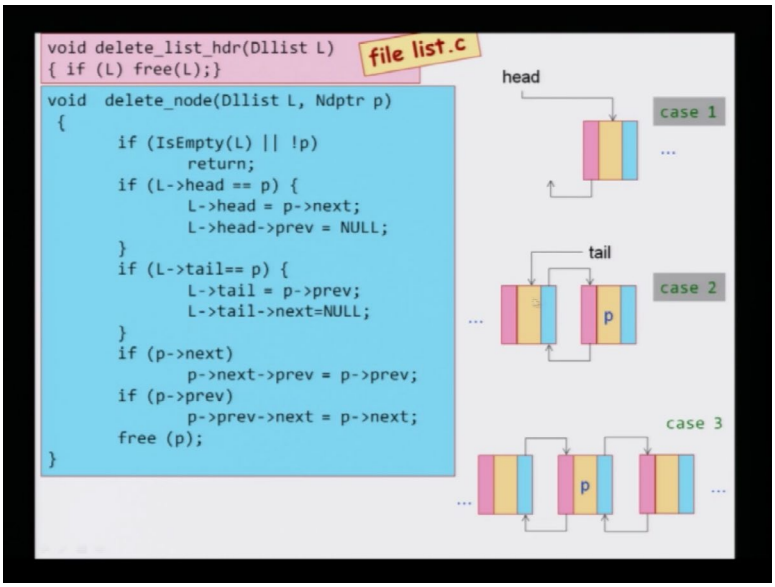
So, head will be made the point to p next. So, this is the line here, L head will go to p next. Now this guys previous will be set to null, because we are going to delete this node.

(Refer Slide Time: 09:26)



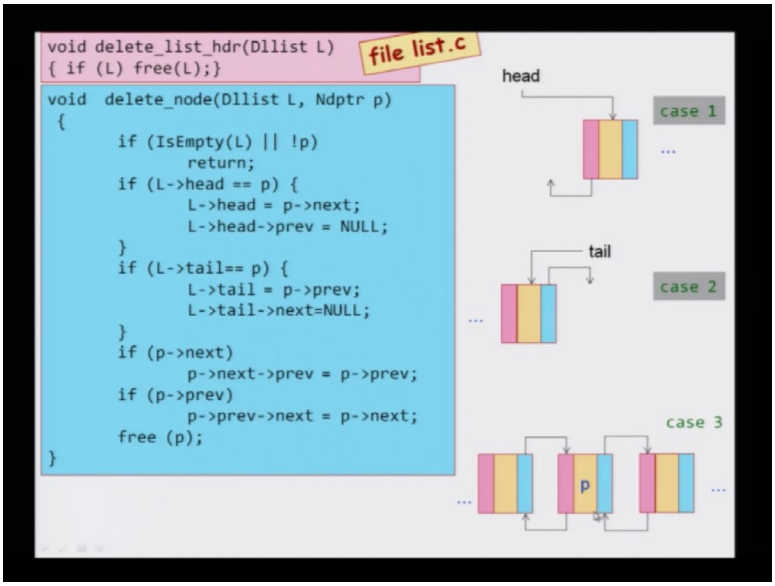
So, this guys previous will be set to null. So, now it does not point anything and then you win free(p). So, this is the first case, where p the node to be deleted was the head of the list. Similarly, if you want to delete the tail of the list. So, now what should you do here, the tail should go to p previous. So, in case two when p is the end of the list that we want to delete.

(Refer Slide Time: 09:52)



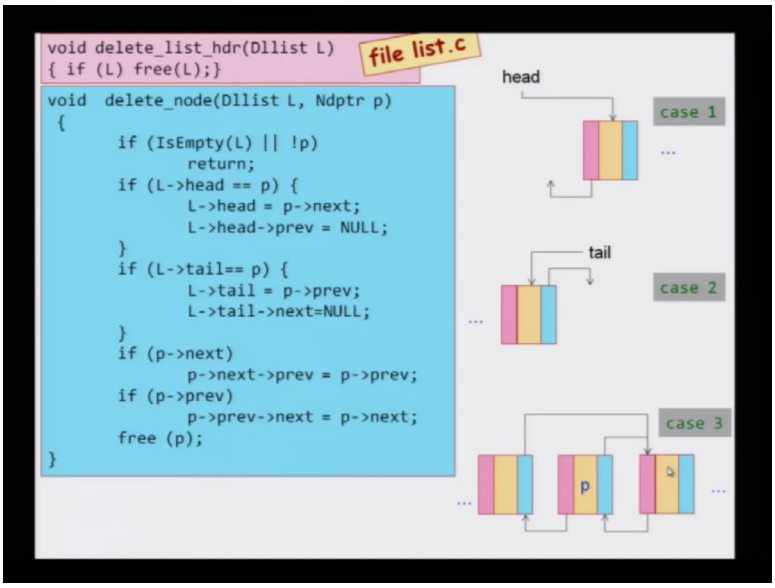
Then tail should go into previous. Now this guys next to will now point to null, because we are going to delete this node.

(Refer Slide Time: 10:02)



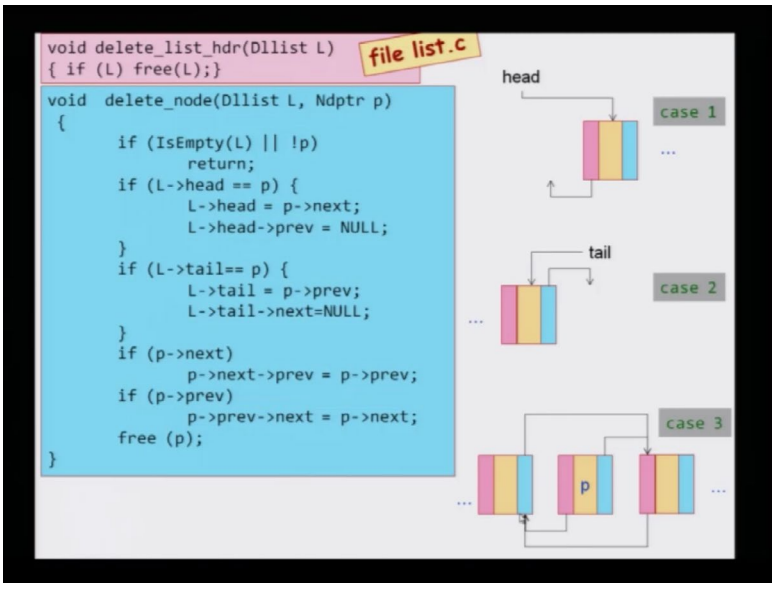
And finally we will free(p). So, L tail will go to p previous, L tail next will be null, and then finally you will free(p). So, we have seen two easy cases; one is delete a head, and other is delete a tail, and now we will see the difficult case where p is an intermediate node. So, in this case what we will do? So, we will we have to remove this node. So, p previous next node should be the next node of p previous. So, this link should point to the node after p. So, that is the first thing.

(Refer Slide Time: 10:45)



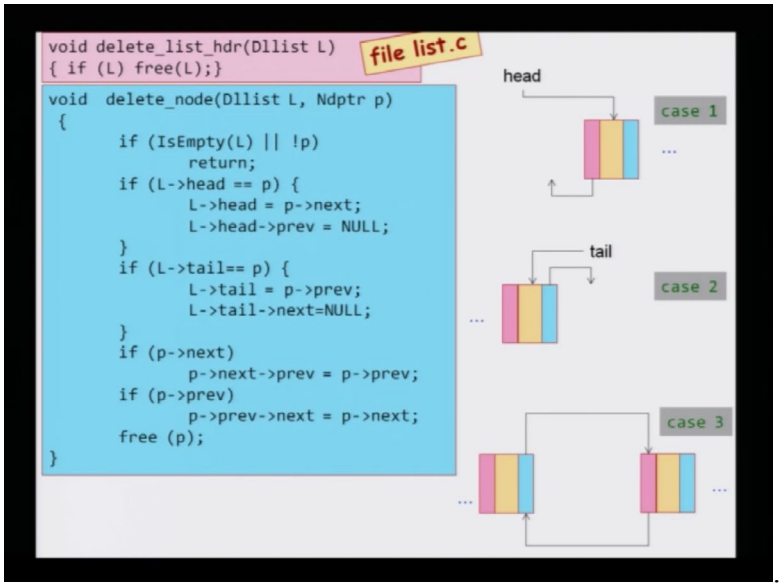
So, we will make this point to the node after p, and this node previous should point to the node before p.

(Refer Slide Time: 10:56)



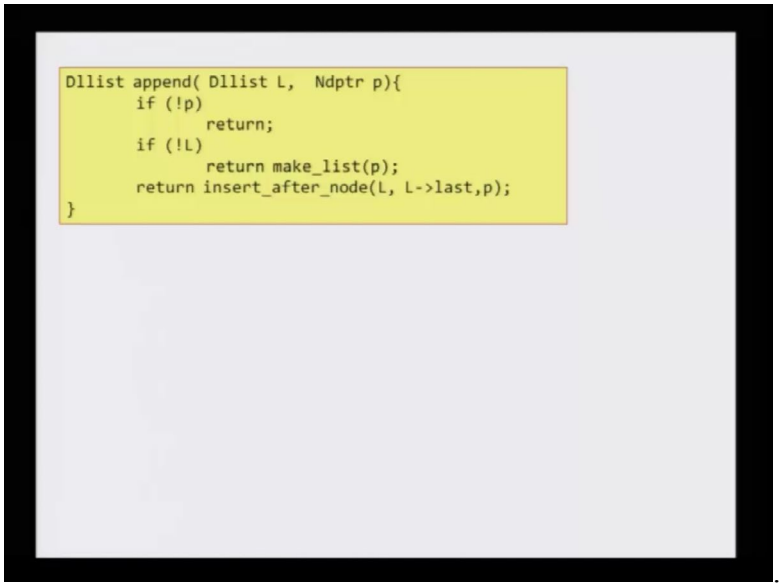
So will reset the links. Now, if you look at the link this guys next is the one after p, this guys previous is the one before p. So, now p can be safely removed.

(Refer Slide Time: 11:09)



So, this is how you would delete a node in the intermediate list. So, if there is the next node, then p next previous will be p previous, that is this backward link. And if there is a the previous node then p previous next will be p next, that is this forward loop. And finally, after that you will free(p). So, this is how you would be delete a node from a link list from a doubly link list, and other operations can be done in a similar manner and some of these operation will be asked in their excise problem that you will be assigned. Similarly, you can think of an extract node, the code will be exactly identical to before except at the end, you will instead of freeing p, you will return p. You do not free the p node, we will just the return the p node.

(Refer Slide Time: 12:08)



Now, let us look at one more example, how do you attend append one node to the end of the list. So, first we will check that the node is pointing to a normal node, if it is pointing to a null node that is nothing to be the done. So, there is nothing to be appended. So, you have return. Now, if there is a list, then what to you do is if there is no need list what you do is you make an list with only one node which is p. Now if there is a list, you can in order to append the node at the end, what you could do is call insert_after_node(L, L->last,p) So, append will be the same as insert the node p at the end of the list. So, you will say insert after L last, what is the node to be inserted p? So, if you have in insert_after_node or an insert before node, you can do this to implement other functions.

So, this is the brief introduction to doubly link list which are similar to singly link list, but facilitate forward as well as backward traveling from a current node, using that you can implement more functions easier than a singly link list. At the same time, it has all the advantages of a singly link list in the sense that if you want to insert a node, it can be done using a constant of number operation, if you want to delete a node it can be done in the constant number of operations. So, those advantages are similar to a singly link list. At the same time the disadvantages are also similar to a singly link list. In the sense that if you want to search through even a shorted doubly link list, you have to search through all the elements.