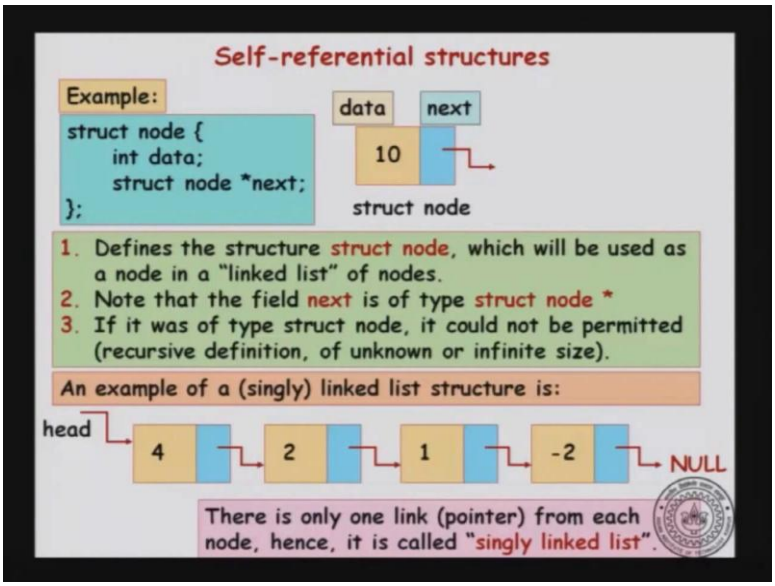
Introduction to Programming in C
Prof. Satyadev Nandakumar
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 51

Once we know structures and pointers to structures, we can introduce some very important data structure called link list. So, we will first see what link list are, how to operate on them, and then argue why link list are useful.

(Refer Slide Time: 00:20)

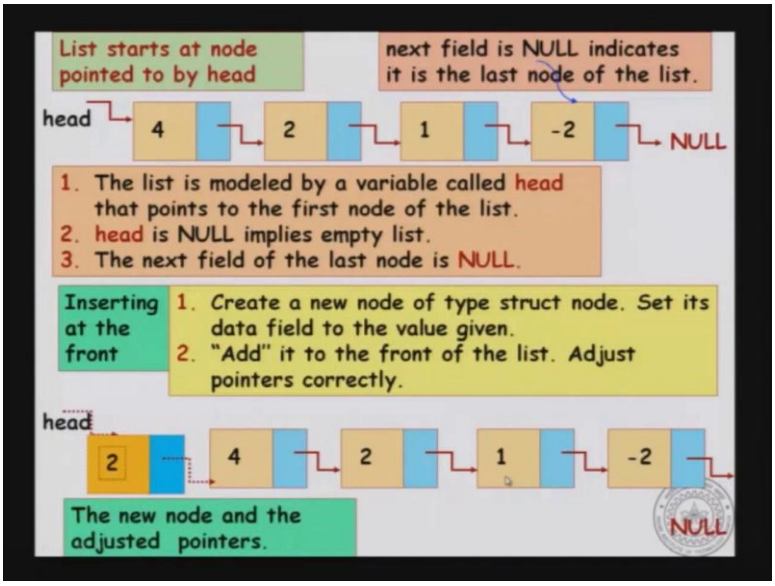


So, let us just introduce this notion called self referential structures. So, we are defining a struct node that has 2 fields - one is an int data, and the next field is the surprising one, it is a pointer 2 type struct node. So, this data structure, this c structure is called a self referential structure because internally there is a pointer to an object of the same type. So, in that sense it refers to some other object of the same type. So, it is called self referential.

So, an example would be like this where the data field has 10, and the next field points to something else which should be a struct node So, then feel next is of type struct node; now, there is a subtle point to be emphasized; instead of struct node star had I return struct node next then this is not allowed, because the definition of struct node has an internal struct node inside it, so which essentially has infinite size. So, we are cleverly avoiding that by including just a pointer to the next node.

So, using this structure we can define what is known as a singly link list. So, an example of a singly link list structure would be where we have pointer which we will call the head of the list, head points to the first struct which is 4, which as data 4; and it is linked to another struct which has data 2, that is linked to another struct which as data 1, and so on. The last struct in the list will be linked to null. So, there is only one link from each node, hence the name singly link list.

(Refer Slide Time: 02:17)

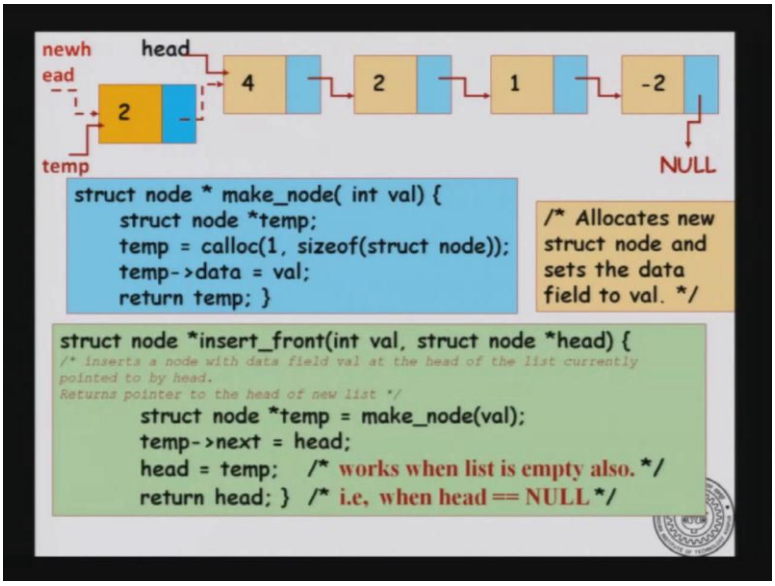


So, the fact that the next field is null indicates that that is the last node in a link list. And a link list is always identified by its head which is the pointer to the first node in the link list. Once we reach the first node we can travel the list by using just the next links. So, once we have a link to 4, we can always say, 4, 2, 1, minus 2, and so on. So, the list is made modeled by the variable called the head that points to the first node in the list. And if the head is null then that means the list is empty. And then you have a bunch nodes; and when once we reach a node with the next field null then that is the last node in the link list.

Now, let us look at certain simple operations on singly link list. Suppose you want to insert a node at the front of the list, so we have a list, 4, 2, 1, minus 2, and we want to insert something else in the beginning. So, what you do is you create a new node of type struct node and set its data field to whatever number that you want to store. Now, add it to the front of the list; we will see how this can be done.

So, suppose that the head is now pointing to 4, and the list is 4, 2, 1, minus 2, and you want to add a new node, so 2 is the node, the data field is 2, how do you add? You do 2 operations – first you say that 2 is next is the first node in the old list. So, that would insert 2 here. And then now the list has changed because the first element is now 2, so head moves to 2; head was previously 4 and head now moves to 2. So, this is abstractly how you would insert a node at the beginning of a list.

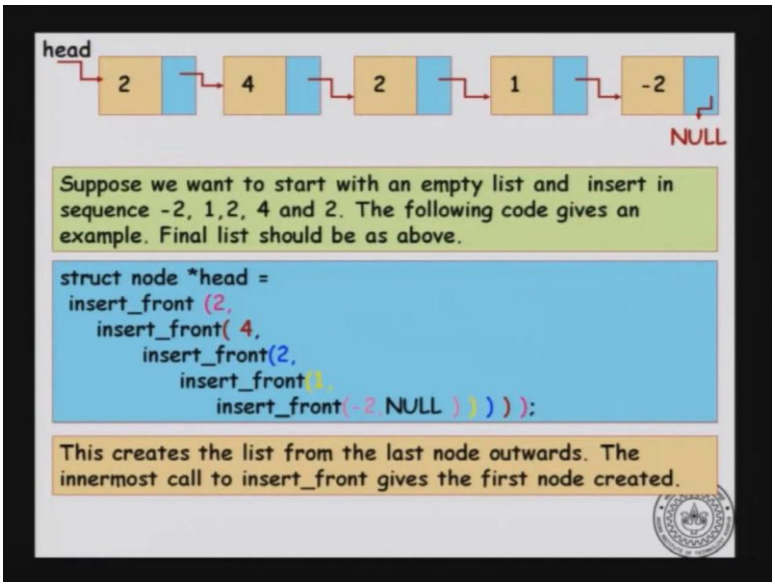
(Refer Slide Time: 04:20)



So, now, let us try to code it and see. So, first we need a code small function to make a node with the given data. So, we will say, struct node star make node in 12. Now, we will create a pointer struct node star temp, then use one of the malloc function called calloc; so 1, size of struct node. So, this will allocate memory enough to create 1 node. Now, this, that memory its data field will be set to val which is what we are given as argument to the function and then you return the node.

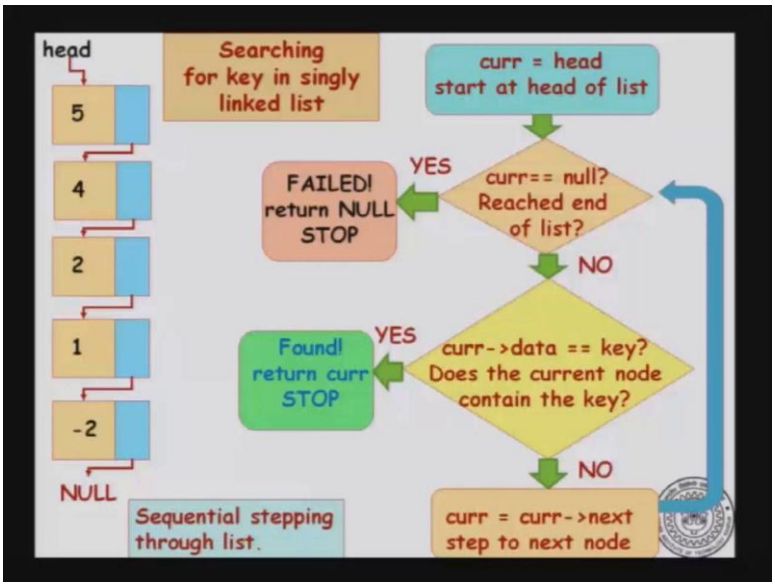
So, we have created a node; and how do you insert in the front? Once you have, once you receive an value to be inserted at the beginning of a list. So, the list is identified by the head. So, we have to create a node which contains the value and insert it at the beginning of this list. So, what we do is we first create a node with the value using make node function; now temp next is set to head; so this link is activated. So, 2s next will be 4. So, that is the first step. The second step is that the head now has to move to 2 because the first element in the new list is 2, not 4. So, I will just say, head equal to temp, and return head which is the head of the new list.

(Refer Slide Time: 06:04)



And now, you can call this function multiple time. Suppose you want to start with an empty list and insert minus 2, then insert minus 1, then insert 2, then insert 4, then insert 2, you can call these functions one after the other. So, I can just say struct node head equal to insert front 2 of, insert front 4 of, insert front 2 of, 1 of minus 2. So, this is the function; this is the sequence of functions and this is the list that you will end up with. So, once you have the function to insert at the beginning of a list, you can use that function multiple times to build up the list.

(Refer Slide Time: 06:46)

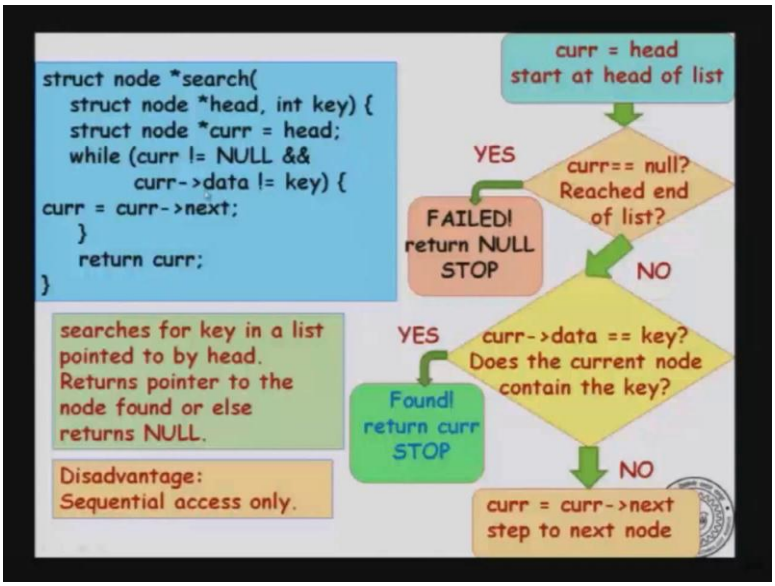


Now, let us look at some useful functions to be done on link list. So, once you have a link list it will be good if you can search the link list to see whether an element is present or not. So, we will look at a very simple algorithm. So, we want to search for a particular key that is an element in a singly link list. So, how do you do it? Abstractly, what you do is you start with the head, see whether the data field in the head node is the key that you want.

If it is, then you are done and say, that I have found it; if it is not what you do is, you go to the next node through the next link and then search; ok, you said the data field of the next node if it is you are done; if it is not, you search and follow the next node. You follow this procedure until you reach the last node. Suppose, you have not found the key even in the last node, and you follow the next link and it is null; so once you reach null then you know that you have reached the end of the list. So, you cannot take null next that will cause your code to crash. So, once you know that your node is null, you can end the search and then say that the key is not present in the list.

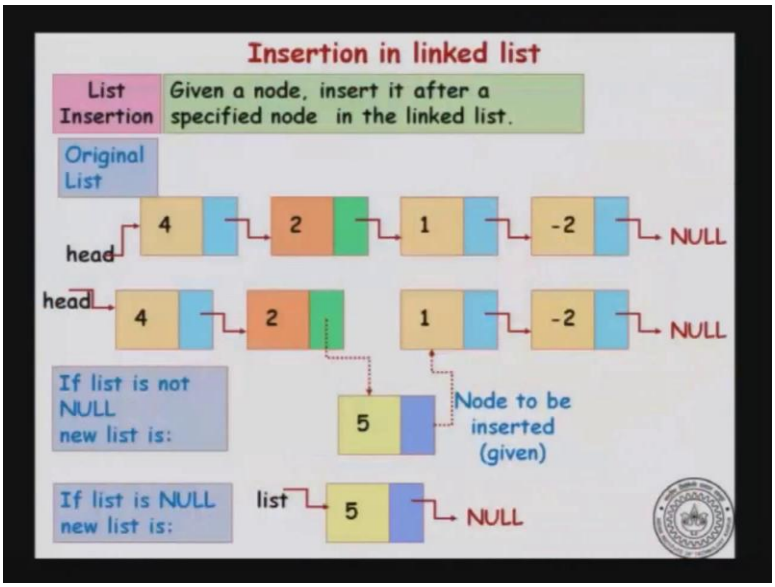
Here is a flowchart corresponding to that; you start with current equal to head; now, is current null, if the current is null then that means you have reached the end of the list and you have not found the key. So, if the current is not null then there is data still to be searched. So, you see whether current data is equal to key; if it is yes then you have found the key, otherwise you follow the next node link to go to the next node in the link list; and again, repeat the procedure, and you can code this in a straight forward manner.

(Refer Slide Time: 08:45)



So, I will write struct node star search; I need the head of the list and I need the key. You start with current equal to head. If current is not equal to null and current data is not equal key, you follow the next link, current equal to current next, and you repeat the procedure. So, when you exit the list either current will be null or current data will be key. So, what is the condition when you reach the return? So, if the key is absent then you are returning null, if the key is present you are returning the pointer to the node, pointer to the first node that contains the key. So, convince yourself that the code works.

(Refer Slide Time: 09:30)

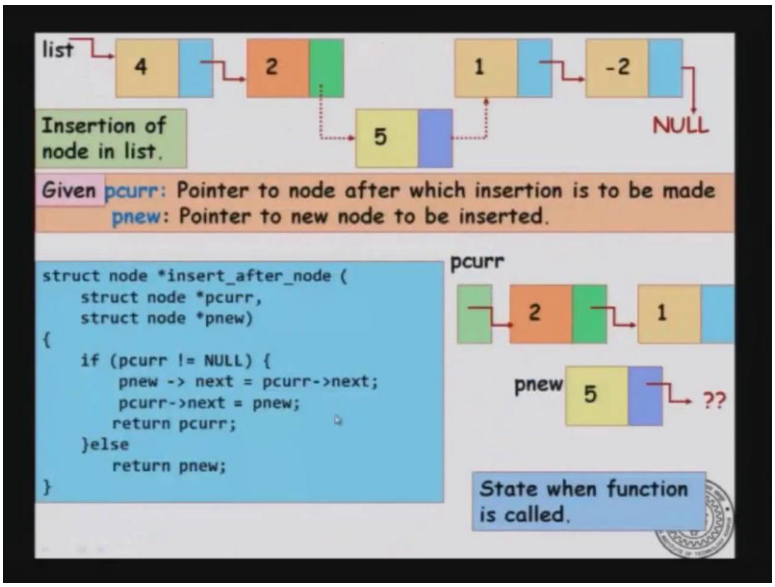


Now, let us look at slightly more involved operations. We have seen insert at the head of the list and that was fairly simple, now suppose you want to insert into the middle of the list; you do not want to insert right at the front, you want to insert somewhere in the middle. Now, there are 2 cases here. If the list is null, that is the easy case; if the list is null then insert in the middle is essentially insert at the front. So, we already have seen the code for that.

Now, if the list is not null, now it is a new algorithm. So, let us look at an example. So, suppose the list is 4, 2, 1, minus 2, and I want to insert a node 5 after node 2. So, how do I do it? 2s next link was 1. And what we need to do in this case is I want to say that I have to insert this node 5; 5s next node will be 1; that is, so think about this as it link in a chain. So, you need to disconnect this link say that 2 is now connected to 5, and 5 is then connected to 1.

Now, the only thing to be noted is that the links have to be detached in a particular sequence. So, first I need to say that 5s next is 1, and then I need to say that 2s next is 5. So, convince yourself that the opposite sequence where I say that 2s next is 5, now your code will has no way to proceed because you have lost the how to traverse from 2 to 1. If you say that 2s next is 5, then 5 has no way to know what was the original next node of 2. So, you have to do it in a particular sequence - 5s next is 1, and then 2s next is 5. We will see this code in a minute.

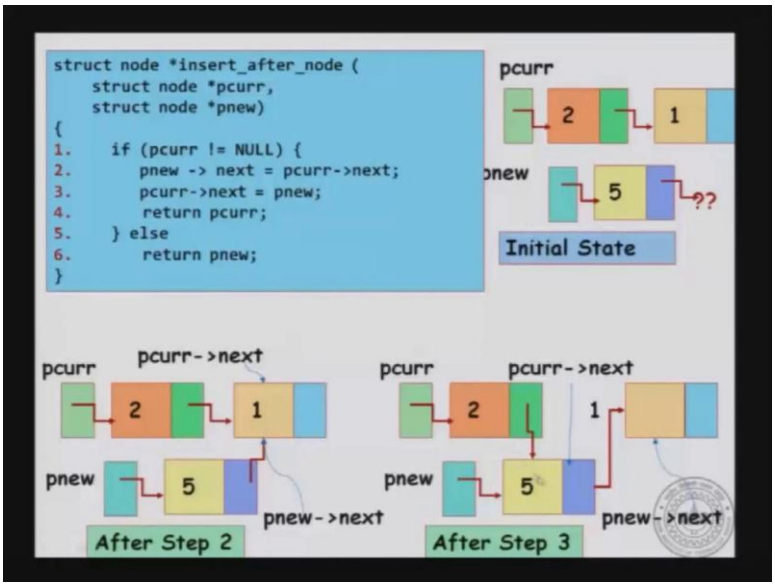
(Refer Slide Time: 11:31)



So, what we do is we want to insert after a node, so p current is the node after which we have to insert; and p new is pointing to the new node that we have to insert. If p current is null, then essentially the list is basically p new; this is a case that we have seen before. If p current is not null, that means the list is not empty, then what you do is, the new node's next node is, p current's next node. So, 5s next node is the old 2s next node which is 1. So, 5 next will be set to 1.

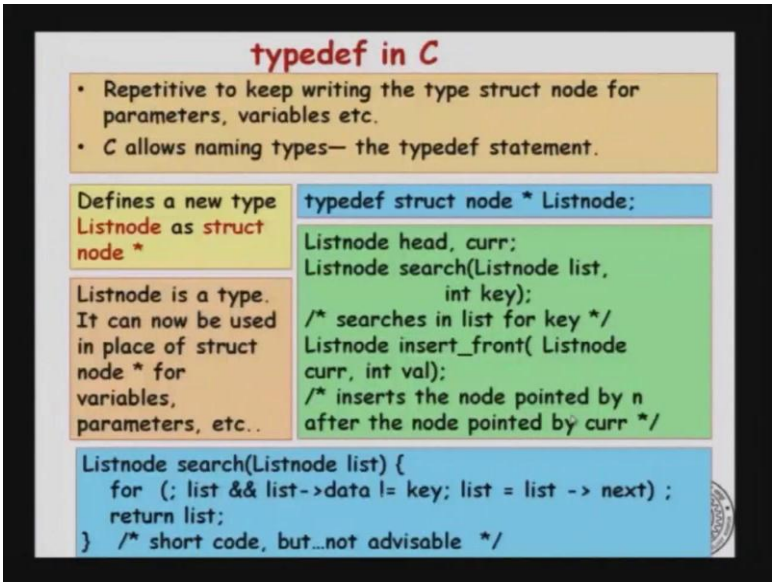
And after that I will say, p current next equal to p new; then I will say that p current which is 2s next will be set to 5. So, convince yourself that if I swap these 2 lines, if I swap the lines order then the code will not work. So, see this for 5 minutes and you convince yourself that that will not work.

(Refer Slide Time: 12:40)



So, let us just see how this works. So, initially, let us say that I want to insert after 2, and new is the new node. So, initial state is something like this; the 5s next node is pointing to something, maybe some arbitrary location. Now, after line 2 that is p new next equal to p current next, this is the state of pointers. Please look very carefully. So, 5s next will point to 1, and then this point 2s next is also pointing to 1. So, there are 2 nodes whose next is 1 which is fine because we have not completely inserted 5 into the list now. Now, at this point, I will just detach 2s next and make it point to 5. So, there we go. So, after step 3 you have essentially inserted 5 into the list.

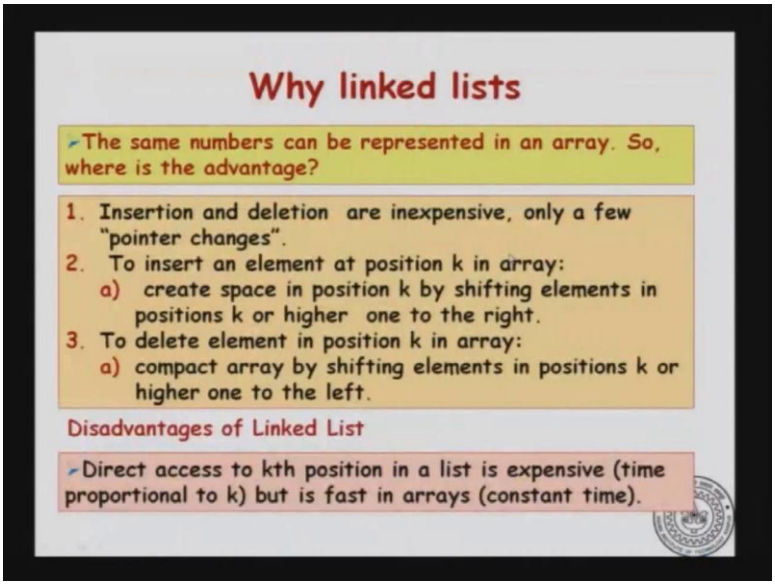
(Refer Slide Time: 13:40)



Now, let us look at some syntactic conveniences that c provides you. So, repetitively you are typing the struct node, and things like that is it is just too much to type, and c allows you to define short names for types. So, if I want to say, like struct node star I want to use the name list node. So, I will just say struct; instead of struct node star head, I will just say list node head. So, it is a shorter way to do it.

So, how do I write this? This is using what is known as a typedef keyword in c. So, if I say, typedef struct node star list node, what it means is that list node is another name for the long type struct node star. So, this is something that you may use if you want to. It is not something that is that you should use, but it is just convenient. So, if I say, list node head, this is the same as saying struct node star head comma current.

(Refer Slide Time: 14:50)

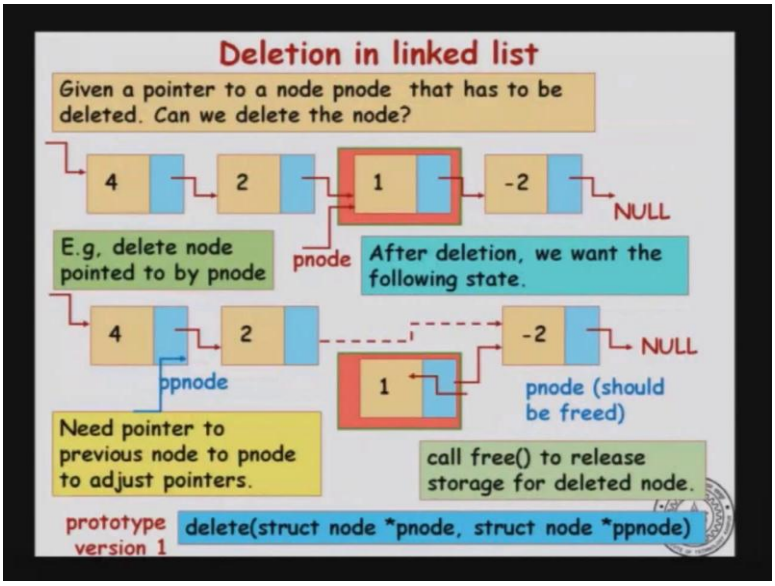


Now, let us see why link lists are important. So, first of all it is one of the first non trivial data structures that you learn. In earlier days when c had only fixed size arrays, link list was important when you needed variable size storage. Nowadays, c has variable size storage, so you can, in arrays, so that is not so important any more. But, here is one important thing, one difference between link list and arrays which are very important.

Like, insertion and deletion in link list are fairly cheap. In the case of an array, so if you want to insert an element at position k in an array, you have to copy all elements from k to n minus 1, to the last element in the array; move each of them backwards, makes space for it, and then insert the k th array. So, this involves, in the worst case, it involves moving all the elements of the array by 1 element, 1 position each.

Similarly, for delete; suppose, you want to delete an element from array, then what you have to do is, you have to take the remaining elements of the array and move them one position to the left. So, this will involve moving n elements in the array if array has n elements in the worst case. Whereas, note that in the link list case, to insert or delete any element, a new node, whether at the beginning or in between, it just takes one operation; the other elements need not be manipulated.

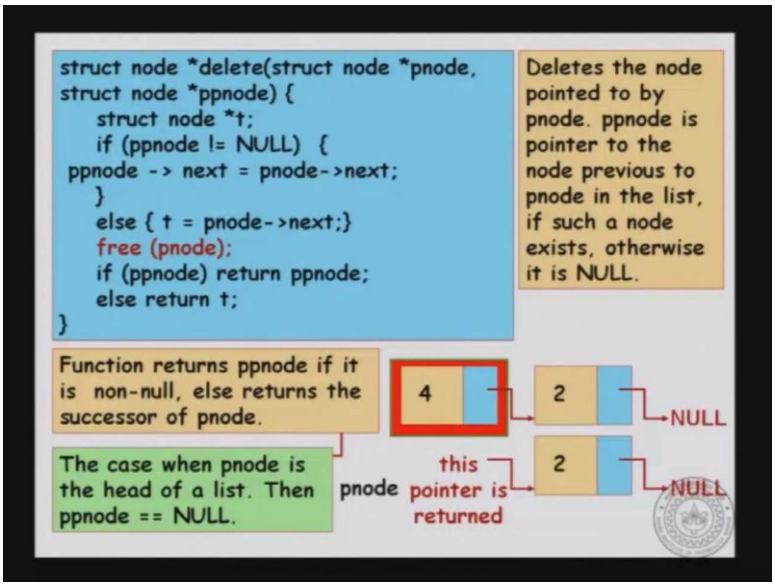
(Refer Slide Time: 16:40)



So, let us just quickly see how to delete a particular node in a link list. So, we will just say that 4, 2, 1, minus 2, is the link list, and I want to delete this particular node, is that possible? So, I cannot straight forward delete this node because the previous nodes next element should point to this guys next element. So, if I want to delete 1, what do I have to do? I have to say that 2s next node should be minus 2.

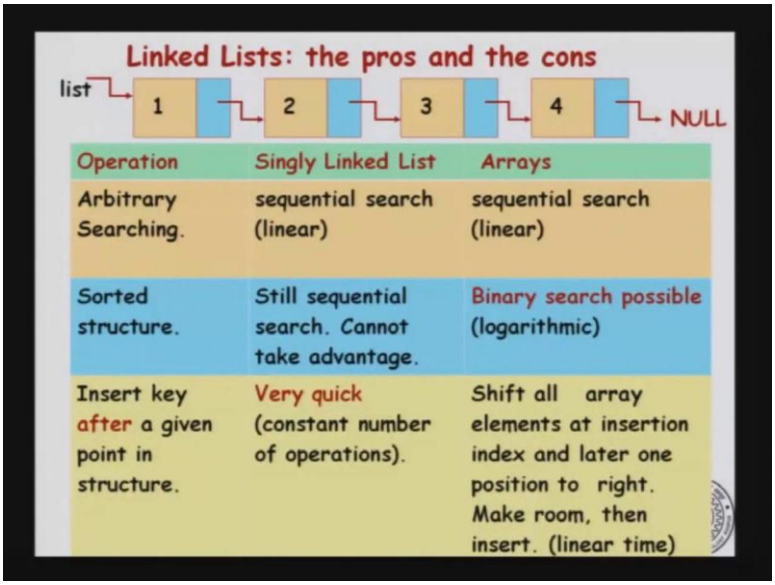
But, in a singly link list there is no way to go back; from 1 you cannot easily get to 2. So, this is slightly; so deletion requires slightly some more information. So, if I can delete a node, if I also have a handle a pointer to its previous node, then it is very easy to say that 2s next node will be minus 2.

(Refer Slide Time: 17:40)



And that is what we will do. So, we will say that let us have a delete function, t node is the node that I want to delete, and pp node is its previous node. And what I will do is, if there is a previous node I will say, previous node's next is the current node's next. So, 2s next link will go to minus 2. If there is no previous node then I will say that, t equal to at the current node's next; and then once that is done, you delete the current node, p node. So, this is how you would delete an element from the link list.

(Refer Slide Time: 18:25)



So, just recap searching in a link list will take order n time, in the case of a link list, that is you have to search all the elements in the worst case which is the same in an array. Now, suppose you sort an array, you have faster search techniques available; you can do binary search in an array. Unfortunately, in a link list, even if you sort the link list, there is no way to do a binary search in the link list; why is this? Because you cannot reach the middle element of the link list in one shot.

In an array, you can just say a mid, and it will go to the… So suppose you say that mid equal to 0 plus n minus 1 by 2, you can go to the middle element of the array. But, there is no way to do that in a link list, you have to go one after the other. So, sorting does not help in searching when you are looking at singly link list. But on the other hand, insertion and deletion are very quick in a link list, whereas they are very slow in an array.