head from the new linked list. They also explain the steps they took to create the new head, such as making ptr equals the first element and making ptr's next equal to head. They use the same process to make p equals the head and make p's next equal to head's next. Finally, they return the head with the new node added to the beginning of the linked list

2.5 Deletion in a Linked List | Deleting a node from Linked List Data Structure

I am going to tell you in a very clear way how to delete any node in a linked list. This method is based on a data structure that has been used to create the list. The algorithm uses a linked data structure to create an algorithm that can be used to delete nodes in a list or delete nodes. It is used for a list to create the list, which includes deleting a link in a link list, or a list with a node that is linked to a list of linked lists or deleting a new node in the linked list, or deleting it in a new list. The algorithm is designed to make a list that can delete nodes and delete nodes, and delete them. This technique involves removing nodes, deleting nodes, creating nodes, adding nodes, removing lists, creating lists, adding lists, taking lists, discarding lists, and removing nodes and destroying lists. After this video, you will never ask how to delete a node in a linked list again.

After this video you'll never ask how to do deletion of any node in linked list because I am going to tell you in a very clear way. If you haven't already accessed my playlist of data structure algorithms, do it now! I do not know why, but I truly appreciate that many of you have shared these videos on Instagram and tagged me. If you want to tag me, feel free to use @codewithharry. When you delete a node in a linked list, you cannot leave its memory on hold, as it is dynamic memory allocated on a dynamic basis. If you do not free it, its memory will be lost. In interviews, candidates are often asked about the allocation and deallocation of memory in linked lists. I have been given the head, which is pointing to the first index, and the index of the node to be deleted (let's assume it's index 2). If you write a

case 2 to delete a node in between, you have to know what to do. I will show you how to delete a node in between nodes in our linked list.

3.1 Single Linked List Operations | Data Structures Tutorial

All operations that can be performed on a single linkedlist can be done using the Malloc or Ml log function in the C language. In new technologies such as Java, there are predefined implementations of these functions available in the collections framework. When inserting elements into the linkedlist, a temporary variable must be created, which is a user-defined data type variable called a node structure type variable. The memory for the node is allocated dynamically using the Malloc function or Ml log function. The values and information that need to be placed inside the node can now be added. It is crucial to determine where the node should be connected based on whether it should be added at the start of the available linkedlist or the end. If the root element does not point to any other elements, the newly created element should be the root element. The linkedlist does not have any elements initially, so all elements must be inserted manually using scanf and the arrow operator to read the link field and data field. This involves providing the base address to the data location address using the temporary variable. The remaining logic for inserting elements must be written in the else block. When inserting an element directly, the temporary variable must be assigned to the root. The temporary variable is a local variable that will be destroyed along with the frame destruction.

Four elements are there for t it's null how to store all these things. I will explain so here it is root element is pointing to root element pointing to the first location first node and some of the nodes already connected. Now we are inserting one more element already node is ready right. For example, using a malloc function. based on null pointer only we should travel from first location to last location. First to p2 link. We have to check what is the pvalue is a 1 0-2 2, 4 1 0, 0-2-4-2 link. This is data field and this is the link field suppose if this link field is a null field means what it is the last node because no other connection but now it is not the null field. This is the insertion part how to insert an element in a linkedlist. For that so we need to declare one local variable another temporary variable by which we

4.2 Double Linked List in Data Structures and Algorithms | Part-2

Passage A covers how to find the length of a linked list, display all its elements, and add a new element to the beginning of the list using a simple logic. It provides a code that demonstrates these techniques, and highlights the use of a temporary variable to allocate memory and read data. However, it does not discuss appending a node in a double linked list, allocating memory with the help of a malloc function, or any techniques used in cinemas. Passage B offers additional information on how to append a node in a double linked list, allocate memory with the help of a malloc function, and some techniques used in cinemas. It highlights the importance of establishing three connections to ensure proper connectivity between nodes and points out that left-side connections cannot be established directly. The passage also mentions using a scanf function to read data and how to determine if a list contains nodes or not based on the root value. Therefore, combining the information in Passage A and B, one can learn how to find the length, display all elements, add a new element to the beginning of a linked list, and append nodes in a double linked list. Additionally, one can learn about the importance of establishing connections between nodes, allocating memory with the help of a malloc function, and using scanf to read data. However, the information on techniques used in cinemas seems unrelated to the topic discussed in Passage A, and therefore, may not be relevant to the learner's objective.

To add a node at the beginning of a linked list, we need to use the malloc function to allocate memory for the new node, as described in passage A. According to passage B, we can also simply write root instead of a temporary variable. Additionally, we can use a while loop to check if the list is empty, or use an "if else" statement. To find the length of the linked list, we can traverse the list and count the number of nodes, as described in passage A. Passage B suggests we can use a while loop to count the nodes, and that we can simply print the number of nodes instead of checking if a node is present or not. To display all the elements in a double linked list, we can traverse the list starting at the root and print each node's value, as described in passage A. Passage B reminds us that we can also use a while loop and print the information of each node using the temp variable or

true. [UNK] "any number you give then it will execute. So we have to give invalid input how all these things just depends on the case you selected" "we are printing. the message very clearly invalid input This two lines gap gap are given after printing that one invalid input" "No still program is executing of course logic not in define".
Felician explains how the ease Felician is important in C language. [UNK] "we are writing prototypes. We are writing so before main functions Generally we will place the prototypes in the declaration of a prototype. No need to write the element just wide push of in teaser sir" when a stack is full there is no place to insert the new element into the stack place is not there so here simply whenever whenever a top value is equals to capacity minus 1. Then we can return 1 1 means what condition true so by that time. Whenever we are trying to push one more element it will give the error message directly is full condition true. when a stack is under flow. Here it is stack is. under flow nothing but no elements in the stack or else. We are just printing that a deleted item, deleted item or. Pop word item is percentage D slash in and here we are printing that item. What item you deleted that item..
We are checking directly is empty or not if it is empty. There were no elements directly. we are printing stack of i. A value is a traversing from 0 to up to top location. All the elements it will print simple. So this is now we have to write prototype of all only. in a switch case. After execution of every case. What we have to write important thing is what break statement or else it will continue with the remaining cases. the program represents how to perform all the stack operations using static arrays nothing but with a constant memory. in the next session. We will discuss how to use a dynamic memory using a static memory..

5.4 Stack Using Single Linked List - Part 1 | Data Structures Tutorial

In this session, we will discuss how to implement a stack using the single linked list concept. To follow the stack rules, we will implement it using single linked list nodes. All the operations would be performed in such a way that it aligns with last in first out (LIFO) and will be performed with the help of a top variable. It is important to mention here that the top variable plays a critical role in stack operations as LIFO is maintained through its

element at q[rare], wrapping around to the beginning of the queue if necessary. If the queue is full, the rare value equals front-1.

Deletion involves moving the front variable forward by one and printing the deleted item. If rare value equals size-1, insertion is not possible. To insert an element, the rare value is incremented by one. If rare equals size, the queue is full. The circular queue allows for efficient insertion, deletion, and display operations.

5.8 Operations on Circular Queue | Data Structures Tutorial

We will focus on circular queues and look at the different operations that we can perform on them. These operations include: Insert an element Delete an element Display the elements First, we need to initialize the circular queue and then we'll perform the aforementioned operations on it. Let's consider the insertion process first. Suppose we have a circular queue, and we want to insert an element into it. Here's how we can do it: Retrieve the position of the element to be inserted, let's say it's at the end of the queue Collect the element from this position Assign this element to a variable, say "deleted_item" Increment the "front" pointer to the next position Now, let's look at the process for deleting an element from the circular queue. Suppose we have a circular queue with front pointer at position 0 and rare pointer at position 2, containing the elements [10, 20, 30, 40, 50, 60]. Here's how we can delete an element: Delete the element pointed by the "front" pointer, which is 30 in this case Increment the "front" pointer to the next position After the deletion, the circular queue will contain the elements [40, 50, 60]. Lastly, we can display the elements of the circular queue as per the requirements of our code.