Shastri 6th Semester

Computer Science

Unit: 3rd

Linked Lists with example

A linked list is a data structure that consists of a sequence of elements called "nodes", where each node contains a reference (or "link") to the next node in the list. The last node in the list typically has a link that points to null, indicating the end of the list.

Here is an example of a simple linked list of integers:

```
struct Node {
   int data;
   struct Node *next;
};
struct Node *head = NULL; // Create an empty list
void append(int value) {
   struct Node *new_node = (struct Node*) malloc(sizeof(struct Node));
   new_node->data = value;
   new_node->next = NULL;
if (head == NULL) {
      head = new_node;
   } else {
      struct Node *current = head;
      while (current->next != NULL) {
         current = current->next;
      }
      current->next = new_node;
```

```
    }
}
```

This example defines a simple linked list of integers, with a structure called "Node" that contains an integer "data" and a pointer "next" to the next node.

The "append" function is used to add new nodes to the list, it creates a new node, assigns the value to its data field, and sets its next field to NULL. Then it checks if the head of the list is null, if so, the new node becomes the head, if not, it goes through the list and it adds the new node to the end of the list.

Linked lists are useful in situations where elements are frequently inserted or removed from the middle of the list, since adding or removing an element from the middle of a linked list requires only updating.

Traversing a linked list, insertion into a linked list, Deletion from a linked list.

Traversing a linked list refers to the process of visiting each node in the list in a sequential manner, typically starting at the head of the list. This can be done using a loop that starts at the head of the list and follows the next pointer of each node until the end of the list is reached.

Insertion into a linked list refers to the process of adding a new node to the list. This can be done in several ways, such as adding a node to the head of the list, the tail of the list, or a specific position in the list. To insert a new node at the head of the list, for example, the new node's next pointer would be set to the current head of the list, and the head pointer would be updated to point to the new node.

Deletion from a linked list refers to the process of removing a node from the list. This can be done in several ways, such as removing a node from the head of the list, the tail of the list, or a specific position in the list. To delete a node from a specific position in the list, the previous node's next pointer would be updated to point to the next node, effectively skipping over the node to be deleted.

A: A linked list is a data structure in C that consists of a sequence of nodes, where each node contains a data element and a pointer to the next node in the list. It is a linear collection of data elements, called nodes, where each node points to the next node in the list. Linked lists are often used to implement dynamic data structures, such as stacks and queues, and are also used in graph algorithms.

A linked list can be implemented in C using a struct type, where each node of the list is represented by an instance of the struct, and each struct contains a data element and a pointer to the next node.

For example, the following code defines a struct called "Node" that represents a node in a singly linked list and contains integer data and a pointer to the next node:

Example

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

Linked lists can be manipulated using various functions such as insertion, deletion, searching, traversing, and so on, which are implemented by manipulating the pointers to the next node.

Some question for practice

1. What is a linked list in C?
2. How do you create a linked list in C?
3. What are the main advantages of using linked lists in C?
4. How do you add a new node to a linked list in C?
5. How do you delete a node from a linked list in C?
6. How do you traverse a linked list in C?
7. How do you reverse a linked list in C?
8. How do you sort a linked list in C?
9. How do you implement a stack using a linked list in C?

10. How do you implement a queue using a linked list in C?
11. How do you implement a circular linked list in C?
12. How do you implement a doubly linked list in C?
13. How do you implement a linked list with a sentinel node in C?
14. How do you handle memory allocation and deallocation in a linked list in C?
15. What are some common errors encountered when working with linked lists in C?