Types of Lists

• Depending on the way in which the links are used to maintain adjacency, several different types of linked lists are possible.

– Linear singly-linked list (or simply linear list)

One we have discussed so far.



– Circular linked list

• The pointer from the last element in the list points back to the first element.



- Doubly linked list
 - Pointers exist between adjacent nodes in both directions.
 - The list can be traversed either forward or backward.



Basic Operations on a List

- Creating a list
- Traversing the list
- Inserting an item in the list
- Deleting an item from the list
- Concatenating two lists into one

List is an Abstract Data Type

- What is an abstract data type?
 - It is a data type defined by the user.
 - Typically more complex than simple data types like *int*, *float*, etc.
- Why abstract?
 - Because details of the implementation are hidden.
 - When you do some operation on the list, say insert an element, you just call a function.
 - Details of how the list is implemented or how the insert function is written is no longer required.

Conceptual Idea



Example: Working with linked list

Consider the structure of a node as follows:

```
struct stud {
    int roll;
    char name[25];
    int age;
    struct stud *next;
  };
```

/* A user-defined data type called "node" */
typedef struct stud node;
node *head;

Creating a List

How to begin?

• To start with, we have to create a node (the first node), and make head point to it.

head = (node *) malloc(sizeof(node));



Contd.

- If there are n number of nodes in the initial linked list:
 - Allocate n records, one by one.
 - Read in the fields of the records.
 - Modify the links of the records so that the chain is formed.



```
node *create_list()
```

{

}

```
int k, n;
node *p, *head;
printf ("\n How many elements to enter?");
scanf ("%d", &n);
for (k=0; k<n; k++)
Ł
    if (k == 0) {
      head = (node *) malloc (sizeof(node));
      p = head;
    }
   else {
           p->next = (node *) malloc (sizeof(node));
           p = p - next;
    scanf ("%d %s %d", &p->roll, p->name, &p->age);
}
p \rightarrow next = NULL;
return (head);
```



node *head; head = create_list();

Traversing the List

What is to be done?

- Once the linked list has been constructed and *head* points to the first node of the list,
 - Follow the pointers.
 - Display the contents of the nodes as they are traversed.
 - Stop when the *next* pointer points to NULL.

```
void display (node *head)
  int count = 1;
  node *p;
 p = head;
  while (p != NULL)
  Ł
    printf ("\nNode %d: %d %s %d", count,
                   p->roll, p->name, p->age);
    count++;
    p = p - > next;
 printf ("\n");
```



node *head;

•••••

display (head);

Inserting a Node in a List

How to do?

- The problem is to insert a node before a specified node.
 - Specified means some value is given for the node (called *key*).
 - In this example, we consider it to be roll.
- Convention followed:
 - If the value of roll is given as negative, the node will be inserted at the end of the list.

Contd.

- When a node is added at the beginning,
 - Only one next pointer needs to be modified.
 - head is made to point to the new node.
 - New node points to the previously first element.
- When a node is added at the end,
 - Two next pointers need to be modified.
 - Last node now points to the new node.
 - New node points to NULL.

• When a node is added in the middle,

- Two next pointers need to be modified.
 - Previous node now points to the new node.
 - New node points to the next node.

```
void insert (node **head)
    int k = 0, rno;
    node *p, *q, *new;
   new = (node *) malloc (sizeof(node));
    printf ("\nEnter data to be inserted: ");
    scanf ("%d %s %d", &new->roll, new->name, &new->age);
    printf ("\nInsert before roll (-ve for end):");
      scanf ("%d", &rno);
   p = *head;
    if (p->roll == rno) /* At the beginning */
       new - next = p;
        *head = new;
```

```
else
```

```
while ((p != NULL) && (p->roll != rno))
      q = p;
      p = p - next;
  }
  if (p == NULL) /* At the end */
      q->next = new;
      new->next = NULL;
 else if (p->roll == rno)
                     /* In the middle */
                                      The pointers
              q->next = new;
                                       q and p
              new->next = p;
                                       always point
          }
                                      to consecutive
                                       nodes.
```



node *head; insert (&head);

Deleting a node from the list

What is to be done?

- Here also we are required to delete a specified node.
 - Say, the node whose roll field is given.
- Here also three conditions arise:
 - Deleting the first node.
 - Deleting the last node.
 - Deleting an intermediate node.

```
void delete (node **head)
{
    int rno;
    node *p, *q;
    printf ("\nDelete for roll: ");
      scanf ("%d", &rno);
    p = *head;
    if (p->roll == rno)
             /* Delete the first element */
    {
        *head = p->next;
        free (p);
    }
```

```
else
      while ((p != NULL) && (p->roll != rno))
      Ł
          q = p;
          p = p - next;
      }
      if (p == NULL) /* Element not found */
         printf ("\nNo match :: deletion failed");
      else if (p->roll == rno)
                    /* Delete any other element */
               q->next = p->next;
               free (p);
```

A sample main() function

```
int main()
```

{

node *head;

```
head = create_list();
display(head);
```

insert(&head); display(head);

```
delete(&head);
display(head);
```

Few Exercises to Try Out

- Write functions to:
 - 1. Concatenate two given lists into one big list. node *concatenate (node *head1, node *head2);
 - 2. Insert an element in a linked list in sorted order. The function will be called for every element to be inserted.

void insert_sorted (node **head, node *element);

3. Always insert elements at one end, and delete elements from the other end (first-in first-out QUEUE).

void insert_q (node **head, node*element)

node *delete_q (node **head) /* Return the deleted node */

More Exercises

- 4. Implement a circular linked list, and write functions to insert, delete, and traverse nodes in the list.
- 5. Represent a polynomial as a linked list, where every node will represent a term of the polynomial $(a_n x^n)$, and will contain the values of 'n' and ' a_n '. Write a function to add two given polynomials.