UNIT-II

REPRESENTATION OF DICTIONARY

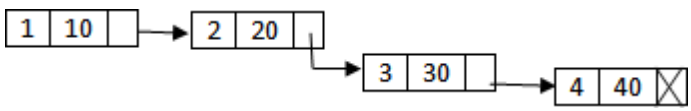
- The dictionary can be represented as a linear list.
- The linear list is a collection of pair and value.
- There are two methods for representing linear list.

1. **Sorted Array** - an array data structure is used to implement the dictionary.
2. **Sorted Chain -** a linked list data structure is used to implement the dictionary.

The insert operation:

- Before any insertion of a node it is first checked if the node is already present in the dictionary or not.
- If it is not present then only the node is inserted in the list.
- There won't be any duplicate keys in the dictionary.
- The contents of dictionary are in the pair form of key and value. The linear list is used to represent such a dictionary.

The linear list can be represented as given below:



- Here the first field represents key and second field represents value.

Structure of linear list for dictionary:

The class definitions for creating dictionary using linear list is as shown below:

class sll

{

```
private:
struct node
{
        int key; int value;
        struct node *next;
}*head; public:
sll();//constructor void insert();
void print(); void del(); int length();
};
```

Operations on this linear list are:

1. Insertion of record in to the list.
2. Deletion of any record from the list.
3. Finding length/size of the list.
4. Display of the list.

Insertion of new node in to the dictionary:

- Initially dictionary will be empty.
- i.e. head=NULL, where head represents the first node in the dictionary.
- We do the insertions in the ascending order of the keys.
- Whenever we want to do an insertion we should know the values to be inserted like, key and value.
- After that we have to create a node with these details (key, value).
- Later we have to place this node in the appropriate position in the list.
- Insertion is same as insertion of Single linked list ay any position.

Example:

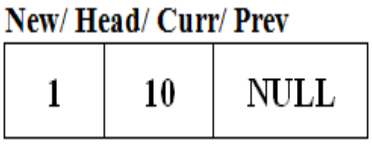
- insert(1,10), (4,20), (7,80) and (3,15).
- First insert (1,10):

create a new node **New** with key=1, value=10

New

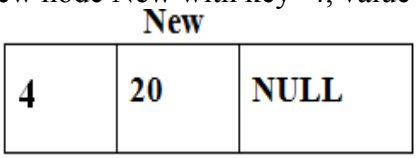
1	10	NULL

As this is the first pair to insert, make this as the **head, curr, prev** node.

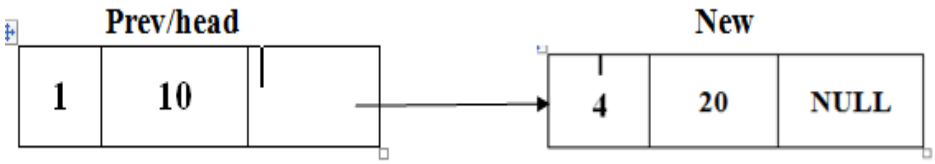


Now insert (4, 20):

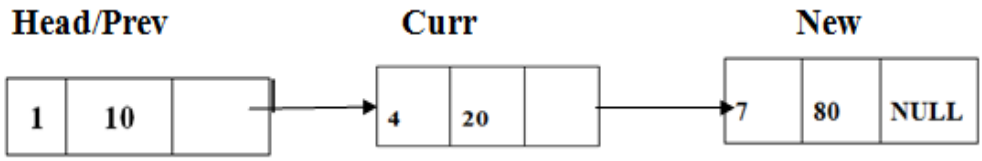
create a new node **New** with key=4, value=20.



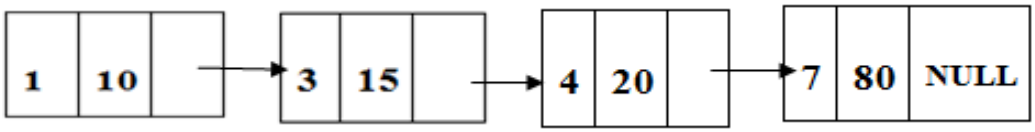
- Place the **New** node in its appropriate position (such that the resultant will be in ascending order).



Insert key = 7, value = 80



- Insert key = 3, value = 15



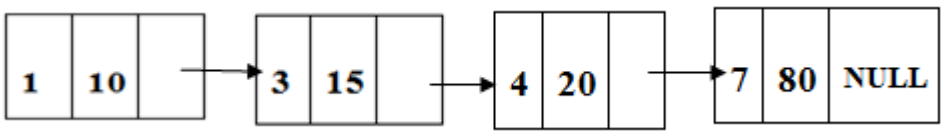
The delete operation:

```
void sll:: delete()
{
        node *curr, *prev; int k;
        curr=head;
        cout<<"\nEnter the key value that you want to delete: "; cin>>k;
        while(curr!=NULL)
                {
                        if(curr->key==k)//traverse till required to delete
                                break;
```

```
                prev=curr;
                    curr=curr->next;
            }
            if(curr==NULL)
                    cout<<"\nNode not found";
            else
            {
                    prev->next=curr->next;
                    //intermediate or end node
                    delete curr;
                    cout<<"\nThe item is deleted\n";
            }
            getch();
}
```

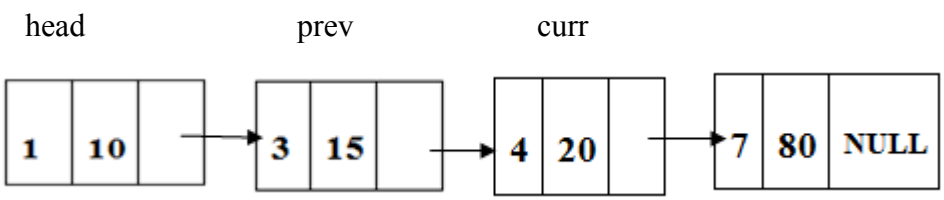
case 1 :

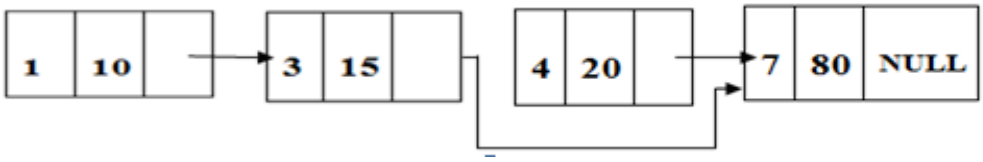
Consider the following list representation of a dictionary,



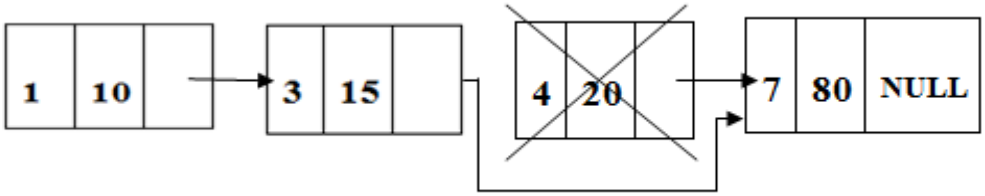
- To delete a node(record) with a given key, start from head node and compare the key value, if the key is not found go to the next node, continue this till we reach to a node with the given value or we reach to a node whose key is greater than the key value or we reach to the end of the list.
- Once we find a node with given key value, mark that node as curr. Now delete this node curr.

Example: Delete a node with key = 4.

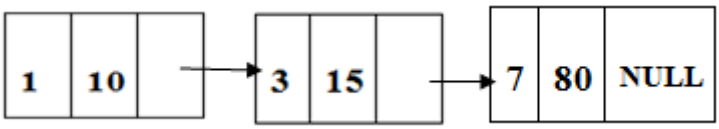




curr



Now the list becomes:

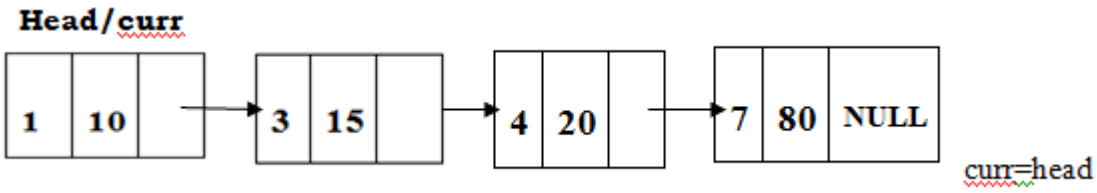


The length operation: // returns number of pairs in the dictionary

```
int sll::length()
{
        node *curr;
        int count = 0; curr=head;
        if(curr==NULL)
        {
        cout<<"The list is empty"; getch();
        return 0;
        }
        while(curr!=NULL)
        {
        count++; curr=curr->next;
        }
        getch();
        return count;
}
```

- By length function the total length/size of linear list is obtained.

Consider the following list



case 1:

if(curr==NULL) i.e list is empty then return 0 as length of list.

case 2:

Otherwise initialize a variable 'count' by zero and as we traverse each node starting from 'head' node, we will increment count by 1. Hence after traversing the complete list, the count variable holds the total number of nodes in the list.

Applications of Dictionaries:

- Database Management Systems - Books catalogue, Employee details
- Compiler - Symbol Table
- Operating System - Page Mapping Tables
- Web Search Engines