* Assignment No. 10 *

- Title: program for find how many maximum compairs may require for finding any keyword. Using Balance (AVL).

- Objective:-
  1) To understand concept of tree.
  2) To understand concept of array structure.

- problem statement:-

A dictionary store keyword & its meaning. provide for facility for adding new keyword. deleting & keywords, Updating values of any entry provide facility to display whole data stored in ascending / Degending order. Also find how many maximum any keyword Use Height balance tree & find the complexity for findling a keyword.

- Outcome:
  i) Implementation of sorting searching
  ii) Analysis working of function

- Theory:

Binary search tree which may sometimes also be called as ordered or sorted binary tree is a node based binary tree data structured.

The right subtree of node contains only no. with greater than nodes key both left &

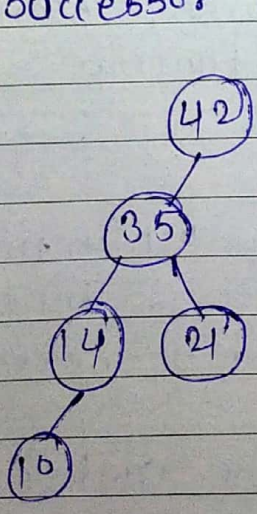
right subtree must also be binary search tree.

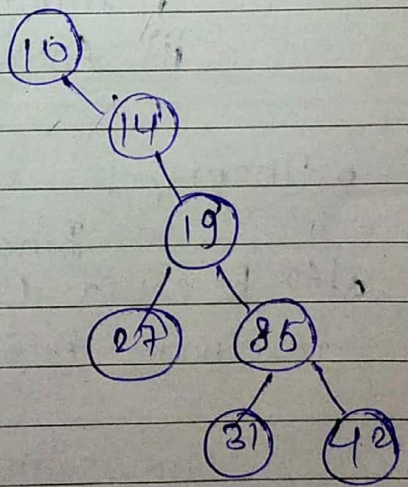
- Insertion:

Insertion begins as a search would begin if root is not right subtree as before eventually we will reach an external node and add value as its right or left child depending on node's values.

```
Void insertnode (node *, &treenode)
{
    node *newnode;
    tree node = newnode;
    else if (node → key, treenode → key);
     insert node (treenode → left, new node)
    else.
    Insert node (tree node → right, newnode)
}
```

Deleting node with two children call the node deleting N. Do not delete N choose successor node.





- strcmp:

strcmp. compare two strings
include <string.h>

int strcmp(const char * RSI. const. char * S)

Description: the strcmp()

Return value:-

Upon completion strcmp() shall return integer greater than equal to or less than 0.

- Algorithm:-

- Create function:-

i) start
ii) Take data from user & create new node n1.
iii) Store data in n1 & make left & right child "NULL".
iv) If root is 'NULL" the assign root to N1.
v) else call insert function & pass root N1
vi) Stop.

- Insert function:-

1) Start
2) we have new node is passed argument.
   so check data in temp is then data in root.
3) If data in root is less than data in temp.
   then check if right child of root is NULL
4) Stop.

- Display function:

1) Start
2) If root is "NULL" then display tree of is not created.
3) otherwise call inorder function & pass root
4) Stop.

- Search function:-

1) Start
2) If not is "NULL" then display tree is not created.
3) Read the key to be updated in variable say k.
4) Search the key to be updated in variable.

- Inorder function:-

1) Start
4) Taking root from inorder function check. if it is NULL or not.

- Inorder function :-
  i) Start.
  ii) Taking root from inorder function check if it is NULL or root.
  iii) Again call inorder function & pass left child Address of node.
  iv) Display data of root.
  v) Again call inorder function.
  vi) Stop.

- main function :

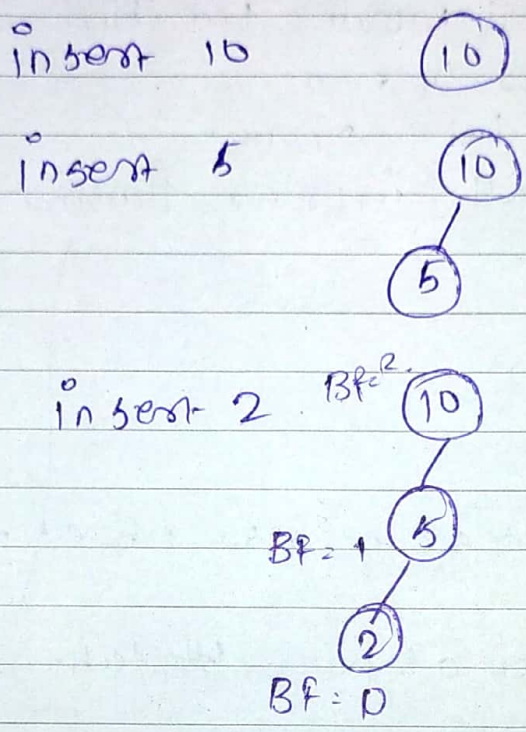
1) Start
2) Create necessary also Object & declare. Variable.
3) Print menu as below ~~take~~ taken choice. User.
4) Create
5) Display
6) Search.
7) update
8) delete
9) Exit.

- Operation :
  Rotations :
  1) LL (Left of left)
  2) RR (Right of Right).
  3) LR (Left of right)
  4) RL (Right of left)_

1) LL :-

Let x be the node with BF equal to +2 after insertion of the new node.

e.g.:- 10, 5, 2.

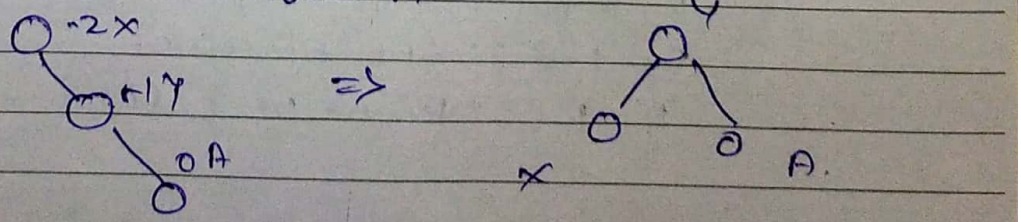


C++ functions for LL :-

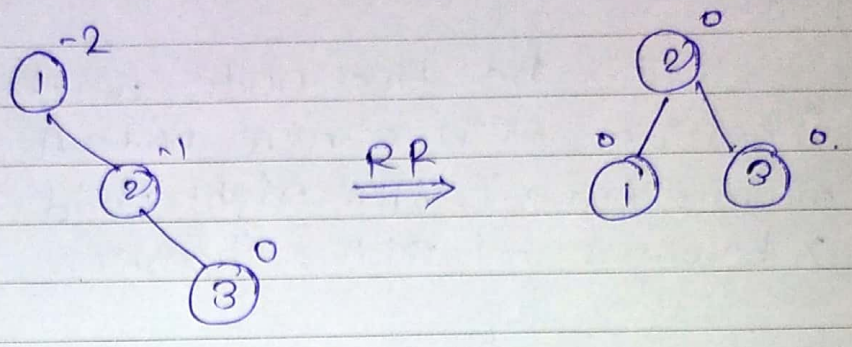
```
node *LL (node *T)
{
    T = rotateright (T);
    return (T);
}
```

2) RR :-

Let x be the with BF. equal to 2. after insertion of the new node A.



e.g.:



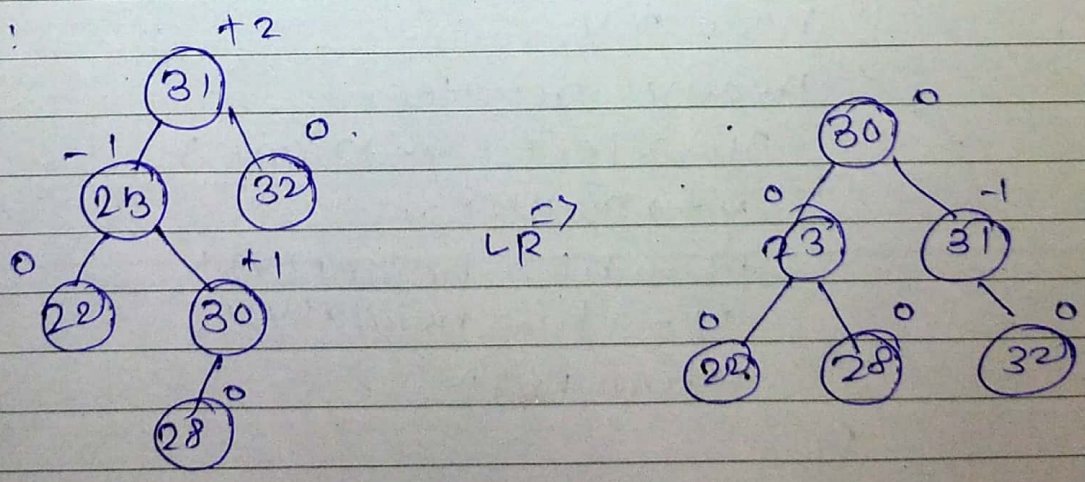
C++ function for RR:

```
node *RR (node *T).
{
    T = rotateleft (T);
    return (T);
}.
```

③ LR -:

Let x be the node with equal to the insertion of the new node A. Balance factor of the node the left child of the node x becomes after insertion of A.

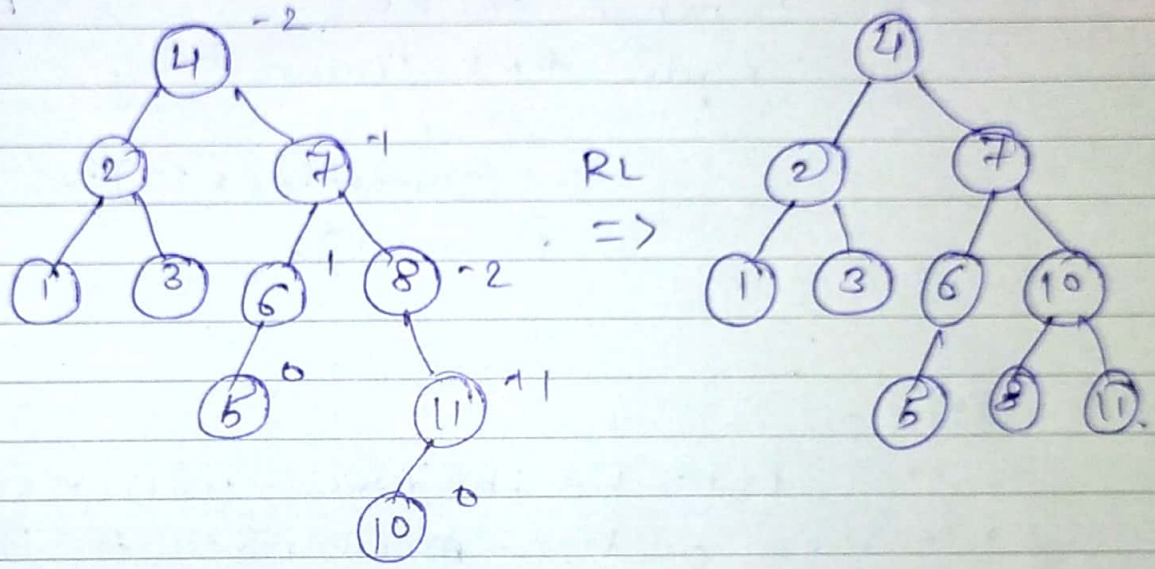
e.g.:



4) RL :-

Let x be the node with Bf = 2 after insertion of the new node A. Balance factor of the node Y, the right child of the node x becomes +1 after insertion of A.

e.g.:




```
node *AVL::rotateright(node *x)
{
    node *y;
    nodey = x->left
    x->left = y->right;
    y->right = x;
    x->ht = height(x);
    y->ht = height(y);
    return(y);
}
```

* Flowchart :-

Start

o/p 1. Create
2. Search
3. print.
4. quit
Enter choice.

switch(op)

case 1: o/p how many nodes you must to insert i/p: n.

Enter data x key. case 2: i=0 i<n → yes i/p i+1 n

search key. case 3: o/p processed or msg. procedure

case 4: stop.

* Conclusion :-

Hence we studied and implemented successfully AVL tree. i.e Height Balance tree.