INPUT:

```
#include<iostream>
using namespace std;
class dict{
   struct node{
      string keyword, meaning;
      int bf;
      node *lc, *rc;
   }*root,*critical;
   int c, cmp;
public:
   dict()
   {
      root=NULL;
   }
   void create(int);
   node* search(string, node*&);
   void balance(string, node*);
   node* rotright(node*);
   node* rotleft(node*);
   void setbf(node *);
   string get_path(node*);
   void update();
   void update_bf(node *);
   void inorder(node *);
   void del(string);
   node *find_max_node(node *);
   void menu();
};

void dict::create(int mode=0)
{
   string path="";
   char ans='y';
   do{
      path="";
      node *curr, *t=new node();
      t->rc=t->lc=NULL;
      t->bf=-999;
      string optional="";
      if(mode==1)
         optional="a new";
      cout<<"\nEnter "<<optional<<" keyword: ";
      cin>>t->keyword;
      cout<<"\nEnter it's meaning: ";
      cin>>t->meaning;
      if(root==NULL)
      {
         root=t;
         setbf(root);
      }
      else
      {
         curr=root;
         while(curr!=NULL)
         {
            if(curr->keyword>t->keyword)
            {
               path+='l';
               if(curr->lc!=NULL)
                  curr=curr->lc;
               else
               {
                  curr->lc=t;
                  setbf(t);
                  break;
               }
            }
            else
            {
               path+='r';
               if(curr->rc!=NULL)
                  curr=curr->rc;
               else
               {
                  curr->rc=t;
                  setbf(t);
                  break;
               }
            }
         }
         c=0;
         update_bf(root);
         //cout<<"\nPath: "<<path;
         balance(path, root);
      }
      if(mode==1)
         break;
      cout<<"\nDo you want to continue?(Y/N): ";
      cin>>ans;
   }while(ans=='y' || ans=='Y');
}

void dict::setbf(node *anynode)
{
   int lcnt=0,rcnt=0;
   node *t=anynode;
   if(t->lc==NULL)
      lcnt=0;
   else
   {
      t=t->lc;
      lcnt++;
      while(true)
      {
         if(t->lc==NULL && t->rc==NULL)
            break;
         if(t->lc==NULL)
            t=t->rc;
         else
            t=t->lc;
         lcnt++;
      }
   }
   t=anynode;
   if(t->rc==NULL)
      rcnt=0;
   else
   {
      t=t->rc;
      rcnt++;
      while(true)
      {
         if(t->lc==NULL && t->rc==NULL)
            break;
         if(t->lc==NULL)
            t=t->rc;
         else
            t=t->lc;
         rcnt++;
      }
   }
   anynode->bf=lcnt-rcnt;
}

void dict::update_bf(node *t)
{
   if(t!=NULL)
   {
      update_bf(t->lc);
      setbf(t);
      if(t->bf<-1 || t->bf>1)
      {
         c++;
         critical=t;
      }
      update_bf(t->rc);
   }
}

/*void dict::get_bf_cnt()
{
   c=0;
```

```
    if(root==NULL)
        return;
    inorder(root);
}
*/
void dict::inorder(node *t)
{
    if(t!=NULL)
    {
        inorder(t->lc);
        cout<<"\nKeyword: "<<t->keyword<<" and meaning: "<<t->meaning;
        inorder(t->rc);
    }
}

dict::node* dict::rotright(node *x)
{
    node *y=x->lc;
    node *b=y->rc;
    y->rc=x;
    x->lc=b;
    return y;
}

dict::node* dict::rotleft(node *x)
{
    node *y=x->rc;
    node *b=y->lc;
    y->lc=x;
    x->rc=b;
    return y;
}

void dict::balance(string path, node* r)
{
    cout<<"\nDisturbed nodes before balance: "<<c;
    if(c==0)
        return;
    int len=path.length();
    if(len<2)
        return;
    node *ggparent,*gparent,*parent,*child;
    ggparent=r;
    gparent=r;
    parent=r;
    child=r;
    string type=path.substr(len-2,2);
    //cout<<endl<<type;
    if(type=="ll")
    {
        for(int i=0; i<len; i++)
```

```
   {
      gparent=parent;
      parent=child;
      if(path[i]=='l')
         child=child->lc;
      else
         child=parent->rc;
   }
    if(gparent==root)
      root=rotright(gparent);
   else
   {
      if(path[len-3]=='r')
         ggparent->rc=rotright(gparent);
      else
         ggparent->lc=rotright(gparent);
   }
}
else if(type=="rr")
{
   for(int i=0; i<len; i++)
   {
      if(path[i]=='l')
      {
         ggparent=gparent;
         gparent=parent;
         parent=child;
         child=child->lc;
      }
      else
      {
         ggparent=gparent;
         gparent=parent;
         parent=child;
         child=parent->rc;
      }
   }
   if(gparent==root)
      root=rotleft(gparent);
   else
   {
      if(path[len-3]=='r')
         ggparent->rc=rotleft(gparent);
      else
         ggparent->lc=rotleft(gparent);
   }
}
else if(type=="lr")
{
   for(int i=0; i<len; i++)
   {
        if(path[i]=='l')
        {
            ggparent=gparent;
            gparent=parent;
            parent=child;
            child=child->lc;
        }
        else
        {
            ggparent=gparent;
            gparent=parent;
            parent=child;
            child=parent->rc;
        }
    }
    parent=rotleft(parent);
    gparent->lc=parent;
    if(gparent==root)
        root=rotright(gparent);
    else
    {
        gparent=rotright(gparent);
        if(path[len-3]=='l')
            ggparent->lc=gparent;
        else
            ggparent->rc=gparent;
    }
}
else if(type.compare("rl")==0)
{
    for(int i=0; i<len; i++)
    {
        ggparent=gparent;
        gparent=parent;
        parent=child;
        if(path[i]=='l')
            child=child->lc;
        else
            child=parent->rc;
    }
    gparent->rc=rotright(parent);
   //cout<<"\nRotate right of rl completed!";
    if(gparent==root)
        root=rotleft(gparent);
    else
    {
        if(path[len-3]=='l')
            ggparent->lc=rotleft(gparent);
        else
            ggparent->rc=rotleft(gparent);
    }
   }
   c=0;
   update_bf(root);
   cout<<"\nDisturbed nodes after balance: "<<c;
}

void dict::update()
{
   string key;
   cout<<"\nEnter keyword to update: ";
   cin>>key;
   del(key);
   create(1);
}
dict::node* dict::search(string k, node *&p)
{
   node *t=root;
   int flag=0;
   cmp=0;
   if(t!=NULL)
   {
      while(t!=NULL)
      {
         cmp++;
         if(k==t->keyword)
         {
            flag=1;
            break;
         }
         else if(k>t->keyword)
         {
            p=t;
            t=t->rc;
         }
         else
         {
            p=t;
            t=t->lc;
         }
      }
      if(flag==1)
         return t;
      else
         return NULL;
   }
   else
      cout<<"\nDictionary empty!";
}

void dict::del(string k)
{
```

```
node *parent=NULL;
node *curr=search(k,parent);
if(curr==NULL)
   return;
if(curr->lc==NULL && curr->rc==NULL)
{
   if(curr!=root)
   {
      if(parent->lc==curr)
         parent->lc=NULL;
      else
         parent->rc=NULL;
   }
   else
      root=NULL;
   delete curr;
}
else if(curr->lc && curr->rc)
{
   node *predec=find_max_node(curr->lc);
   curr->keyword=predec->keyword;
   curr->meaning=predec->meaning;
   del(predec->keyword);
}
else
{
   node *child=(curr->lc)? curr->lc:curr->rc;
   if(curr!=root)
   {
      if(parent->lc==curr)
         parent->lc=child;
      else
         parent->rc=child;
   }
   else
      root=child;
   delete curr;
}
critical=NULL;
update_bf(root);
if(critical!=NULL)
{
   if(k>critical->keyword)
   {
      string path="l";
      path+=get_path(critical->lc);
      balance(path,critical);
   }
   else
   {
      string path="r";
```

```
            path+=get_path(critical->rc);
            balance(path,critical);
        }
    }
}

string dict::get_path(node *temp)
{
    node *t=temp;
    string x,x1="";
    x1=x;
    while(true)
    {
        if(t->lc==NULL && t->rc==NULL)
            break;
        if(t->lc)
        {
            t=t->lc;
            x+='l';
        }
        else
        {
            t=t->rc;
            x+='r';
        }
    }
    t=temp;
    while(true)
    {
        if(t->lc==NULL && t->rc==NULL)
            break;
        if(t->rc)
        {
            t=t->rc;
            x1+='r';
        }
        else
        {
            t=t->lc;
            x1+='l';
        }
    }
    if(x1.length()>x.length())
        return x1;
    return x;
}

dict::node* dict::find_max_node(dict::node* t)
{
    while(t->rc!=NULL)
        t=t->rc;
```

```
   return t;
}

void dict::menu()
{
   int ch;
   string key;
   node*t=NULL;
   do{
   cout<<"\n************AVL Tree Dictionary**************";
   cout<<"\n\t1.Create/Insert keyword";
   cout<<"\n\t2.Search by Keyword";
   cout<<"\n\t3.Display in ascending.";
   cout<<"\n\t4.Delete an entry";
   cout<<"\n\t5.Modify an entry";
   cout<<"\n\t6.Exit";
   cout<<"\nEnter your choice: ";
   cin>>ch;
   switch(ch)
   {
         case 1: create();
               break;
         case 2: cout<<"\nEnter keyword to search: ";
               cin>>key;
               t=search(key,t);
               if(t){
                  cout<<"\nMeaning of "<<t->keyword<<" is "<<t->meaning;
                  cout<<"\nNo. of Comparisons: "<<cmp;
               }
               else
                  cout<<"\nKeyword not found!";
               break;
         case 3: inorder(root);
               break;
         case 4: cout<<"\nEnter keyword to delete: ";
               cin>>key;
               del(key);
               break;
         case 5: update();
               break;
         case 6: cout<<"\nProgram ends!";
               return;
         default: cout<<"\nEntered invalid choice number, try again... ";
                break;
   }
   }while(true);
}

int main()
{
   dict obj;
   obj.menu();
   return 0;
}
```

OUTPUT:

```
************AVL Tree Dictionary**************
       1.Create/Insert keyword
       2.Search by Keyword
       3.Display in ascending.
       4.Delete an entry
       5.Modify an entry
       6.Exit
Enter your choice: 1

Enter  keyword: Monik

Enter it's meaning: Advice

Do you want to continue?(Y/N): Y

Enter  keyword: Sanjhari

Enter it's meaning: Sunshine

Disturbed nodes: 0
Do you want to continue?(Y/N): Y

Enter  keyword: Divyam

Enter it's meaning: Cool

Disturbed nodes: 0
Do you want to continue?(Y/N): Y

Enter  keyword: Hasti

Enter it's meaning: Happy

Disturbed nodes: 0
Do you want to continue?(Y/N): Y

Enter  keyword: Flash

Enter it's meaning: speed

Disturbed nodes before balance: 2
Disturbed nodes after balance: 0
Do you want to continue?(Y/N): N

************AVL Tree Dictionary**************
       1.Create/Insert keyword
```

```
        2.Search by Keyword
        3.Display in ascending.
        4.Delete an entry
        5.Modify an entry
        6.Exit
Enter your choice: 2

Enter keyword to search: Flash

Meaning of Flash is speed
No. of Comparisons: 2
************AVL Tree Dictionary**************
        1.Create/Insert keyword
        2.Search by Keyword
        3.Display in ascending.
        4.Delete an entry
        5.Modify an entry
        6.Exit
Enter your choice: 3

Keyword: Divyam and meaning: Cool
Keyword: Flash and meaning: speed
Keyword: Hasti and meaning: Happy
Keyword: Monik and meaning: Advice
Keyword: Sanjhari and meaning: Sunshine
************AVL Tree Dictionary**************
        1.Create/Insert keyword
        2.Search by Keyword
        3.Display in ascending.
        4.Delete an entry
        5.Modify an entry
        6.Exit
Enter your choice: 4

Enter keyword to delete: Sanjhari

Disturbed nodes before balance: 1
Disturbed nodes after balance: 0
************AVL Tree Dictionary**************
        1.Create/Insert keyword
        2.Search by Keyword
        3.Display in ascending.
        4.Delete an entry
        5.Modify an entry
        6.Exit
Enter your choice: 2

Enter keyword to search: Flash

Meaning of Flash is speed
No. of Comparisons: 1
```

```
************AVL Tree Dictionary**************
        1.Create/Insert keyword
        2.Search by Keyword
        3.Display in ascending.
        4.Delete an entry
        5.Modify an entry
        6.Exit
Enter your choice: 5

Enter keyword to update: Divyam

Disturbed nodes before balance: 1
Disturbed nodes after balance: 0
Enter a new keyword: Sanjhari

Enter it's meaning: Shine

Disturbed nodes: 0
************AVL Tree Dictionary**************
        1.Create/Insert keyword
        2.Search by Keyword
        3.Display in ascending.
        4.Delete an entry
        5.Modify an entry
        6.Exit
Enter your choice: 3

Keyword: Flash and meaning: speed
Keyword: Hasti and meaning: Happy
Keyword: Monik and meaning: Advice
Keyword: Sanjhari and meaning: Shine
************AVL Tree Dictionary**************
        1.Create/Insert keyword
        2.Search by Keyword
        3.Display in ascending.
        4.Delete an entry
        5.Modify an entry
        6.Exit
Enter your choice: 6

Program ends!
```