Rehashing technique:

#include<bits/stdc++.h>

using namespace std;

class HashedObj{

 public:

 int val;

 HashedObj(int x){

 val = x;

 }

};

template <typename HashedObj>

class HashTable{

 public: vector<pair<HashedObj,string>> table;

 int cnt=1,capacity; //maintaining table's count and size for load factor calculation

 auto& MakeTable( int size )

 {

 capacity=nextPrime(size);

 for(int i = 0; i < capacity; i++){ //pair maintains an object and its status

 pair<int,string> newpair;

 table.push\_back(make\_pair(HashedObj(0),"EMPTY")); //creation of vector of pairs

 }

 return table;

 }

 int rem( HashedObj & x ) {

 int pos = x.val% capacity; //searching for the element to delete

 if (table[pos].second == "EMPTY")

 return false;

 else

 {

 for (int j = 0; j < capacity; j++)

 {

 int t = (pos + j ) % capacity;

 if (table[t].first.val== x.val)

 {

 table[t].second="EMPTY";

 break;

 }

 }

 return true;

 }

 }

 void display(){

 for (int i = 0; i < table.size(); i++)

 {

 if(table[i].second=="ACTIVE")

 cout <<i<<": "<<table[i].first.val <<"\n"; //.first.val gives value of element

 //.second gives status of the position

 }

 }

 void rehash(){

 int pos;

 auto &new\_table=MakeTable(2\*capacity); //rehashing with double the size of previous table

 for (int i = 0; i < table.size(); i++)

 {

 if(table[i].second=="ACTIVE")

 {

 pos = table[i].first.val% capacity;

 if (new\_table[pos].second == "EMPTY"){

 new\_table[pos].first.val = table[i].first.val;

 new\_table[pos].second ="ACTIVE";

 }

 else

 {

 for (int j = 1; j < table.size(); j++)

 {

 int t = (pos + j ) % table.size();

 if (new\_table[t].second == "EMPTY")

 {

 new\_table[t].first = table[i].first.val;

 new\_table[t].second = "ACTIVE";

 }

 }

 }

 }

 table=new\_table;

 }

 }

 bool isPrime( int n ){

 if( n == 2 || n == 3 )

 return true;

 if( n == 1 || n % 2 == 0 )

 return false;

 for( int i = 3; i \* i <= n; i += 2 )

 if( n % i == 0 )

 return false;

 return true;

 }

 int nextPrime( int n ){

 if( n % 2 == 0 )

 ++n;

 for( ; !isPrime( n ); n += 2 )

 ;

 return n;

 }

 void insrt(HashedObj & x){

 if(((double)cnt/capacity)\*100>=70){

 rehash();

 cout<<"Table is rehashed, new size= "<<capacity<<endl;

 }

 int pos = x.val% table.size();

 if(table[pos].first.val==x.val){ //If duplicate is found then, prompt an alert

 cout<<"Duplicate element\n";

 return;

 }

 else if (table[pos].second == "EMPTY"){

 table[pos].first.val = x.val;

 table[pos].second ="ACTIVE";

 cnt++;

 }

 else

 {

 for (int j = 1; j < table.size(); j++)

 {

 int t = (pos + j ) % table.size();

 if (table[t].second == "EMPTY")

 {

 table[t].first = x.val;

 table[t].second = "ACTIVE";

 cnt++;

 break;

 }

 }

 }

 }

};

int main(){

 int n;

 cout<<"Decide tablesize: ";

 cin>>n; //reading table size from user

 cout<<"\n";

 HashTable <HashedObj> tab;

 auto &table=tab.MakeTable(n);

 int choice,element;

 do{

 cout<<"1.insertion"<<endl;

 cout<<"2.deletion"<<endl;

 cout<<"3.display"<<endl;

 cout<<"4.Exit"<<endl;

 cin>>choice;

 switch(choice){

 case 1:

 {

 cout<<"enter element: ";

 cin>>element;

 HashedObj ob(element);

 tab.insrt(ob);

 }

 break;

 case 2:

 {

 cout<<"enter element for deletion: "<<endl;

 cin>>element;

 HashedObj ob(element);

 if(tab.rem(ob))

 cout<<"deleted\n";

 else

 cout<<"Element not there\n";

 }

 break;

 case 3:

 tab.display();

 break;

 }

 }while(choice<=3);

 return 0;

}

 bool rem( HashedObj & x ) {

 int pos = ((a\*(x.val)+b)%p)%table.size(); //Universal hashing function

 /\*table size should be very large such that

 collision factor α = 1/table size \*/

 if (table[pos].second == "EMPTY")

 return false;

 else if(table[pos].first.val== x.val)

 {

 table[pos].second="EMPTY";

 return true;

 }

 else

 return false;

 }

 bool isPrime( int n ){

 if( n == 2 || n == 3 )

 return true;

 if( n == 1 || n % 2 == 0 )

 return false;

 for( int i = 3; i \* i <= n; i += 2 )

 if( n % i == 0 )

 return false;

 return true;

 }

 int nextPrime( int n ){

 if( n % 2 == 0 )

 ++n;

 for( ; !isPrime( n ); n += 2 )

 ;

 return n;

 }

 void display(){

 for (int i = 0; i < table.size(); i++)

 {

 if(table[i].second=="ACTIVE")

 cout <<i<<": "<<table[i].first.val <<"\n";

 }

 }

 void insrt(HashedObj & x){

 int pos = ((a\*(x.val)+b)%p)%table.size();

 if (table[pos].second == "EMPTY"){

 table[pos].first.val = x.val;

 table[pos].second ="ACTIVE";

 }

 }

};

int main(){

 int n;

 cout<<"Decide tablesize: ";

 cin>>n;

 cout<<"\n";

 HashTable <HashedObj> tab(n);

 int choice,element;

 do{

 cout<<"1.insertion"<<endl;

 cout<<"2.deletion"<<endl;

 cout<<"3.display"<<endl;

 cout<<"4.Exit"<<endl;

 cin>>choice;

 switch(choice){

 case 1:

 {

 cout<<"enter element: ";

 cin>>element;

 HashedObj ob(element);

 tab.insrt(ob);

 }

 break;

 case 2:

 {

 cout<<"enter element for deletion: "<<endl;

 cin>>element;

 HashedObj ob(element);

 if(tab.rem(ob))

 cout<<"deleted\n";

 else

 cout<<"Element not there\n";

 }

 break;

 case 3:

 tab.display();

 break;

 }

 }while(choice<=3);

 return 0;

}