//A simple example of hashtable

#include <iostream>

#include <cstdlib>

#include <cstring>

#include <iomanip>

#define SIZE\_KEY       16

#define SIZE\_VALUE1    64

#define SIZE\_VALUE2    16

#define DEFAULT\_TABLESIZE    101

using namespace std;

struct NODE

{

   Node(const char\* Key1 = "\0", const char\* fName = "\0",

         const char \*tele ="\0", const double sal = 0.0 )

   {

      strcpy(Key, Key1);

      strcpy(FullName, fName);

      strcpy(Tele\_No, tele);

      Salary = sal;

      Tax = 0.005 \* Salary;

      next = NULL;

   }

   char Key[SIZE\_KEY];

   char FullName[SIZE\_VALUE1];

   char Tele\_No[SIZE\_VALUE2];

   double Salary;

   double Tax;

   Node \*next;

};

class Hashtable

{

   private:

      int table\_size;

      NODE\*\* table;

      int size;

      long hashString(char\* Key);

      NODE\* find(char\* Key);

      NODE\* current\_entry;

      int current\_index;

   public:

      Hashtable(int T = DEFAULT\_TABLESIZE);//constructor

      virtual ~Hashtable();//destructor

      bool put(NODE \*);

      bool get(NODE \*);

      bool contains(char\* Key);

      bool remove(char\* Key);

      void removeAll();

      int getSize();

      void initIterator();

      bool hasNext();

      void getNextKey(char\* Key);

      friend void disp(NODE \*);

};

Hashtable::Hashtable(int T)

{

   size = 0;

   table\_size = T;

   table = new NODE\*[table\_size];

   for(int i=0; i<table\_size; i++)

   {

      table[i] = NULL;

   }

}

Hashtable::~Hashtable()

{

   removeAll();

   delete[] table;

}

void disp(NODE \*N1)

{

 cout << "\nKey:      " << N1->Key << "\nFullName: "

      << N1->FullName <<"\nTele.:    " << N1->Tele\_No << "\nSalary:   "

      << setprecision(12) << N1->Salary<< "\nTax:      " << N1->Tax << endl;

}

bool Hashtable::put(NODE \*N)

{//start put

   if(find(N->Key) != NULL)

   {

      return false;

   }

   NODE\* entry = new NODE(N->Key, N->FullName,N->Tele\_No, N->Salary);

   int bucket = hashString(N->Key);

   entry->next = table[bucket];

   table[bucket] = entry;

   size++;

   return true;

}//end put

bool Hashtable::get(NODE\* N)

{//start get

   NODE\* temp = find(N->Key);

   if(temp == NULL)

   {

      N->FullName[0] = '\0';

      return false;

   }

   else

   {

      strcpy(N->FullName, temp->FullName);

      strcpy(N->Tele\_No, temp->Tele\_No);

      N->Salary = temp->Salary;

      N->Tax = temp->Tax;

      return true;

   }

}//end get

bool Hashtable::contains(char\* Key)

{//start contains

   if(find(Key) == NULL)

   {

      return false;

   }

   else

   {

      return true;

   }

}//end contains

bool Hashtable::remove(char\* Key)

{//start remove

   int bucket = hashString(Key);

   NODE\* temp = table[bucket];

   if(temp == NULL)

   {

      return false;

   }

   else if(strcmp(Key, temp->Key) == 0)

   {

      table[bucket] = temp->next;

      delete temp;

      size--;

      return true;

   }

   else

   {

      NODE\* temp\_next = temp->next;

      while(temp\_next != NULL)

      {

         if(strcmp(Key, temp\_next->Key) == 0)

         {

            temp->next = temp\_next->next;

            delete temp\_next;

            size--;

            return true;

         }

         temp = temp->next;

         temp\_next = temp\_next->next;

      }

   }

   return false;

}//end remove

void Hashtable::removeAll()

{//start removeAll

   for(int i=0; i<table\_size; i++)

   {

      NODE\* temp = table[i];

      while(temp != NULL)

      {

         NODE\* next = temp->next;

         disp(temp);

         delete temp;

         temp = next;

      }

   }

   size = 0;

}//end removeAll

int Hashtable::getSize()

{

   return size;

}

NODE\* Hashtable::find(char\* Key)

{ //start find

   int bucket = hashString(Key);

   NODE\* temp = table[bucket];

   while(temp != NULL)

   {

      if(strcmp(Key, temp->Key) == 0)

      {

         return temp;

      }

      temp = temp->next;

   }

   return NULL;

}//end find

long Hashtable::hashString(char\* Key)

{//start hashString

   int n = strlen(Key);

   long h = 0;

   for(int i=0; i<n; i++)

   {

      //To get almost fair distributions of nodes over the array

      h = (h << 3) ^ Key[i];

   }

    return abs(h % table\_size );

}//end hashString

void Hashtable::initIterator()

{//start initIterator

   current\_entry = NULL;

   current\_index = table\_size;

   for(int i=0; i<table\_size; i++)

   {

      if(table[i] == NULL)

      {

          continue;

      }

      else

      {

         current\_entry = table[i];

         current\_index = i;

         break;

      }

   }

}//end initIterator

bool Hashtable::hasNext()

{

   if(current\_entry == NULL)

   {

      return false;

   }

   else

   {

      return true;

   }

}

void Hashtable::getNextKey(char\* Key)

{

   if(current\_entry == NULL)

   {

      Key[0] = '\0';

      return;

   }

   strcpy(Key, current\_entry->Key);

   if(current\_entry->next != NULL)

   {

      current\_entry = current\_entry->next;

   }

   else

   {

     for(int i=current\_index+1; i<table\_size; i++)

     {

        if(table[i] == NULL)

        {

           continue;

        }

        current\_entry = table[i];

        current\_index = i;

        return;

     }

     current\_entry = NULL;

     current\_index = table\_size;

   }

}

void dispAll(Hashtable\* hashtable);

int main()

{

   char temp1[SIZE\_KEY];

   Hashtable\* hashtable = new Hashtable();

   NODE N1("389","Mariam","8216734",22123.267);

   if(!hashtable->contains(N1.Key))

   {

      cout << "\nAdding node:  ";

      disp(&N1);

      hashtable->put(&N1);

   }

   strcpy(N1.Key, "314");

   strcpy(N1.FullName, "Zeki");

   strcpy(N1.Tele\_No, "8765623");

   N1.Salary = 98124.567;

   if(!hashtable->contains(N1.Key))

   {

      cout << "\nAdding node:  ";

      disp(&N1);

      hashtable->put(&N1);

   }

   strcpy(N1.Key, "320");

   strcpy(N1.FullName, "Murad");

   strcpy(N1.Tele\_No, "7231144");

   N1.Salary = 19834.575;

   if(!hashtable->contains(N1.Key))

   {

      cout << "\nAdding node:  ";

      disp(&N1);

      hashtable->put(&N1);

   }

   strcpy(N1.Key, "768");

   strcpy(N1.FullName, "Hassan");

   strcpy(N1.Tele\_No, "7689876");

   N1.Salary = 45124.755;

   if(!hashtable->contains(N1.Key))

   {

      cout << "\nAdding node:  ";

      disp(&N1);

      hashtable->put(&N1);

   }

   strcpy(N1.Key, "756");

   strcpy(N1.FullName, "Ali");

   strcpy(N1.Tele\_No, "9874545");

   N1.Salary = 43554.125;

   if(!hashtable->contains(N1.Key))

   {

      cout << "\nAdding node:  ";

      disp(&N1);

      hashtable->put(&N1);

   }

   dispAll(hashtable);

    strcpy(temp1,"314");

   hashtable->remove(temp1);

   cout << "\n\nAfter removing 314:" << endl;

   dispAll(hashtable);

   cout << "\n\nDestroying hashtable:" << endl;

   delete hashtable;

   return 0;

}

void dispAll(Hashtable \*hashtable)

{

        NODE N1;

   cout << "\n\nCurrent nodes in hashtable:" << endl;

   hashtable->initIterator();

   while(hashtable->hasNext())

   {

      hashtable->getNextKey(N1.Key);

      hashtable->get(&N1);

      disp(&N1);

   }

}