```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>

#define STUDENT_ID_SIZE 4
#define NAME_SIZE 41
#define DEPARTMENT_CODE_SIZE 4
#define INITIAL_TABLE_SIZE 11


struct student
{
    char studentID[STUDENT_ID_SIZE];
    char studentName[NAME_SIZE];
    char departmentCode[DEPARTMENT_CODE_SIZE];

};

struct student *initializeHashTable(int);
struct student *initializeRehashTable(struct student *studentInfo,int,int,int);

int roundPrime(int);
int findKey(char ID[]);
int hashFunction(char ID[], int);
int quadricProbing(char[],int,int);
int doubleHashing(char[],int,int);

void addStudent(struct student *studentInfo,int,int);
void searchStudent(struct student *studentInfo,int,int);
void printTable(struct student *studentInfo,int);

void openAddressing_Menu();
void showFunctionalities_Menu();


int main()
{

struct student *studentInfo;
studentInfo=initializeHashTable(INITIAL_TABLE_SIZE);

if(studentInfo==NULL)
    printf("Allocation failed!");

int SFchoice;//Shortcut of Show Functionalities
int OAchoice;//Shortcut of Open Addressing
int numberOfStudents=0;//this value hold for #of students in my hashTable
int tableSize=INITIAL_TABLE_SIZE;//this value holds for size of my hash table,which is initialized as 11
float loadFactor;

do//in this loop, I take the open addressing choice from the user
{
    openAddressing_Menu();
    scanf("%d",&OAchoice);
    if(OAchoice!=1 && OAchoice!=2)
        printf("\nPlease enter option as 1 or 2!\n\n");

}while(OAchoice!=1 && OAchoice!=2);


do//in this loop, I take the showFunctionalities choice from the user
{

    do // If user did not enter an option between 1-4, I print error message
    {
        showFunctionalities_Menu();
        scanf("%d",&SFchoice);
        if(SFchoice!=1 && SFchoice!=2 && SFchoice!=3 && SFchoice!=4)
            printf("\nPlease enter options between 1-4!\n");

    }while(SFchoice!=1 && SFchoice!=2 && SFchoice!=3 && SFchoice!=4);


    loadFactor=(numberOfStudents)/(float)tableSize;//this is the load factor, which equals to The total number of students in a hash table / the size of the hash table

    if(loadFactor>0.5)//if it becomes greater than 0.5, I do enter in this if condition and do necessary rehashing function calls
    {
        int newTableSize;//this value holds for my newTableSize(becomes 23 after first rehashing)
        int oldTableSize;//this value holds for my oldTableSize(11, as given initially)
        oldTableSize=tableSize;//I equaled my oldTableSize with table size for not losing the oldTableSize
        //I used oldTableSize in traversing my oldHashTable in initializeRehashTable

        newTableSize=tableSize*2;//here, I multiply the old size with 2
        newTableSize=roundPrime(newTableSize);//Then I called roundPrime function to holds my size with the closest upcoming prime number as a hashTableSize.

        tableSize=newTableSize;//after getting the new prime table size, I set the value of tableSize to this new prime table size.Then I called initializeRehashTable

        studentInfo=initializeRehashTable(studentInfo,tableSize,oldTableSize,OAchoice);//this function initialize new hashTable with its new size
    }

    if(SFchoice==1)
    {
        addStudent(studentInfo,tableSize,OAchoice);
        numberOfStudents = numberOfStudents + 1;//when I added students successfully, I increment #ofStudents by 1
    }
    else if(SFchoice==2)
    {
        searchStudent(studentInfo,tableSize,OAchoice);
    }
    else if(SFchoice==3)
    {
        printTable(studentInfo,tableSize);
    }
    else if(SFchoice==4)
    {
        printf("Exit!");
    }
    else
    {
        printf("Please enter options between 1-4!\n");
    }

}while(SFchoice!=4);


free(studentInfo);

    return 0;
}

struct student *initializeHashTable(int tableSize)//this function initialized a hashTable with given size 11 at the beginning.
{
    struct student *studentInfo;
    studentInfo=(struct student*)malloc(tableSize*sizeof(struct student));
    int i;

    for(i=0;i<tableSize;i++)
    {
        strcpy(studentInfo[i].studentID,"");
        strcpy(studentInfo[i].studentName,"");
        strcpy(studentInfo[i].departmentCode,"");
    }

    return studentInfo;
}

int roundPrime(int newTableSize)//this function simply rounds the doubled table size to closest prime number(for
example: 20 -> 23)
{
    int i,flag=0;

    for(i=2;i<=newTableSize/2;++i)
    {
        if (newTableSize%i==0)//if the newTableSize is not prime, flag will become 1
        {
            flag=1;
            break;
        }
    }

    if (flag==0)//if flag does not change, that means my newTableSize is a prime number, so I can use it as my
new hash table size
    {
        return newTableSize;
    }
    else//if flag is 1,which means newTableSize is not a prime number, I call the function again with
incrementing the table size by 1
    {
        newTableSize++;
        roundPrime(newTableSize);
    }
}

//In this function,I created a new hashTable(copyTable) with the size of newTableSize, and I initialized its
components with "".
//Then I checked my old hashTable(studentInfo) to take the student information in it, then I add the students to
my new copy hastTable(copyTable)
struct student *initializeRehashTable(struct student *studentInfo,int newTableSize,int oldTableSize,int choice)
{
    struct student *copyTable;//here,I defined and give memory to my newHashTable
    copyTable=(struct student*)malloc(newTableSize*sizeof(struct student));

    int i,j;
    char ID[STUDENT_ID_SIZE];
    char studentName[NAME_SIZE];
    char depCode[DEPARTMENT_CODE_SIZE];
    int index,compare;

    for(i=0;i<newTableSize;i++)//here, I initialized my copyTable with ""
    {
        strcpy(copyTable[i].studentID,"");
        strcpy(copyTable[i].studentName,"");
        strcpy(copyTable[i].departmentCode,"");
    }

    for(i=0;i<oldTableSize;i++)//this loop checks which indexes I have an student in my
oldHashTable(studentInfo)
    {
        compare=strcmp(studentInfo[i].studentID,"");//This is where I checked whether given an index in
studentInfo is empty or not
```

```
        if(compare!=0)//If it is not empty(""),that means I have a student in this index, so I copied the values in this index to my parameters.
        {
            strcpy(ID,studentInfo[i].studentID);
            strcpy(studentName,studentInfo[i].studentName);
            strcpy(depCode,studentInfo[i].departmentCode);

            for(j=0;j<newTableSize;j++)//this loop simply adds values, I have achieved from my oldHashTable,to my newHashTable
            {
                if(choice==1)//Choice holds for openAddressing.If it is chosen as 1, I used doubleHashing,otherwise I used quadricProbing
                    index=doubleHashing(ID,newTableSize,j);
                else
                    index=quadricProbing(ID,newTableSize,j);

                compare=strcmp(copyTable[index].studentID,"");//If there exist no collision(given index has values as ""),I copied the information to my newTable
                if(compare==0)
                {
                    strcpy(copyTable[index].studentID,ID);
                    strcpy(copyTable[index].studentName,studentName);
                    strcpy(copyTable[index].departmentCode,depCode);
                    break;
                }
            }
        }
    }

    free(studentInfo);//When I am done, I free the studentInfo,and return to my newHashTable(copyTable)

    return copyTable;
}

void addStudent(struct student *studentInfo, int tableSize,int choice)
{

    int compare,i,index,flag=0;
    char ID[STUDENT_ID_SIZE];

    printf("\nEnter unique identifier:");
    scanf("%s",ID); getchar();

    for(i=0;i<tableSize;i++)
    {
        compare=strcmp(ID,studentInfo[i].studentID);
        if(compare==0)//If the id is not unique, then the application should print "Id should be unique!" and go back to the functionalities menu
        {
            printf("ID should be unique!\n");
            flag=1;//I set the flag 1 and break, so I did not entered the below for loop, and did not ask for the rest of the info of student to user
            break;
        }
    }

    if(flag==0)//if it is 0 as initial, I will take action to add the new student to list
    {
        for(i=0;i<tableSize;i++)//here, I am counting the value of i. If there exist any collision, I increment the i value
        {
            if(choice==1)
                index=doubleHashing(ID,tableSize,i);//I get the index of the student, who will be added to list, by calling the doubleHashing function
            else if(choice==2)
                index=quadricProbing(ID,tableSize,i);
            compare=strcmp(studentInfo[index].studentID,"");//here I check whether the index that I tried to put student record in hash table is empty or not.
            if(compare==0) // If the index is available to put student record, that means no collision, then I added the student to hash table
            {
                strcpy(studentInfo[index].studentID,ID);
                break;//and I break the loop
            }
        }
        //After taking ID, I take the other information of student, and I increment the numberOfStudents(size) that specifies how many students I have in my hash table
        printf("Enter name:");
        gets(studentInfo[index].studentName);
        printf("Enter department:");
        scanf("%s",studentInfo[index].departmentCode);
        printf("%s has been registered!\n",studentInfo[index].studentName);
    }
}

//The search operation will ask for a student id and then try to find the student in the hash table and then print their details if found.
void searchStudent(struct student *studentInfo,int tableSize,int choice)
{
    //index holds for the index of the student in the hash table
    //Flag indicates whether the search is successful or not.Initially, it equals to 0
    //Compare indicates whether the entered ID matches with same ID in hash table
    //Choice holds for the open addressing choice
    int index,i,compare,flag=0;
    char ID[STUDENT_ID_SIZE];

    printf("Enter unique identifier:");
    scanf("%s",ID);

    for(i=0;i<tableSize;i++)//by looking the user choice, if it is 1, I called doubleHashing, otherwise I called quadricProbing to find the index value.
    {
        if(choice==1)
            index= doubleHashing(ID,tableSize,i);// I got the index by calling the doubleHashing function
        else if(choice==2)
            index= quadricProbing(ID,tableSize,i);// I got the index by calling the quadricProbing function

        compare=strcmp(studentInfo[index].studentID,ID);//I compare whether the ID entered by user matches with the ID of the student in hash table
        if(compare==0)//If they match, it means compare is 0, and I print the information of the student in this specific index in my hash table
        {
            printf("\nName: %s\nDepartment: %s\n",studentInfo[index].studentName,studentInfo[index].departmentCode);
            flag=1;//I set the flag as 1, If I found a student, then I break the loop
            break;
        }
    }

    if(flag==0)//If student is not found, then flag remains as 0, and I print the result of failed search
        printf("\nStudent is not found!\n");
}

void printTable(struct student *studentInfo,int tableSize)//This function will simply print the contents of the hash table in the order they are placed.
{

   int i;

   printf("Index\t\tID\t\tName\t\t\tDepartment\t\n");
   for(i=0;i<tableSize;i++)
   {
       printf("%d\t\t%s\t\t%s\t\t%s\n",i,studentInfo[i].studentID,studentInfo[i].studentName,studentInfo[i].departmentCode);
   }

}

int findKey(char ID[])
{
    return (ID[0])-65+ID[1]-48+ID[2]-48;//this is the given formula to find the integer value of the student ID
}

int hashFunction(char ID[],int tableSize) // hashFunction
{
    int key=findKey(ID);
    return (2*key)%tableSize;//hash1(key) function
}


int doubleHashing(char ID[],int tableSize,int i)
{
    int key= findKey(ID);//Key is the integer value of the student ID
    int hashKey=hashFunction(ID,tableSize);//hashKey holds for hash1(key). I used this value in my hash function for double hashing
    return (hashKey+(i*(7-(key % 7))))%tableSize;//hash function for double hashing. i*(7-(key % 7)) holds for f(i), which is hash2(key) function in this case
}

int quadricProbing(char ID[],int tableSize,int i)
{
    int key = findKey(ID);//Key is the integer value of the student ID
    int hashKey=hashFunction(ID,tableSize);//hashKey holds for hash1(key). I used this value in my hash function for quadric probing
    return (hashKey+(i*i)) % tableSize;//hash function for quadric probing. i*i holds for f(i), which is square of i in this case
}

void showFunctionalities_Menu()//This menu shows the 4 options for user
{
    printf("\n1)Add a student\n");
    printf("2)Search a student\n");
    printf("3)Print table\n");
    printf("4)Exit\n");
    printf("What would you like to do(1-4)?:");
}

void openAddressing_Menu()//This menu shows the open addressing options at the beginning
{
    printf("1)Double Hashing\n");
    printf("2)Quadric Probing\n");
    printf("Which open addressing method do you choose(1-2)?:");
}
```