Hash table

a) open addressing
   - linear probing,
   - qudratic probing,
   - double hashing.

b) separate chaining.

Example 1 (C++). Implementation of the hash table using **linear search (probing)**. Variable keysPerCell defines the compression ratio (in the program is equal to 10. This means if the size of the array is 20, entered may be the keys from 0 to 200).

[hash.cpp]

```
//hash.cpp
//demonstrates hash table with linear probing
#include <iostream>
#include <vector>
#include <cstdlib>                      //for random numbers
#include <ctime>                        //for random numbers
using namespace std;

class DataItem
{                                       //(could have more data)
    public:
        int iData;                      //data item (key)

        DataItem(int ii) : iData(ii)       //constructor
        {  }

};  //end class DataItem
//-----------------------------------------------------------------
class HashTable
{
    private:
        vector<DataItem*> hashArray;    //vector holds hash table
        int arraySize;
        DataItem* pNonItem;             //for deleted items

    public:
        HashTable(int size) : arraySize(size)  //constructor
        {
            arraySize = size;
            hashArray.resize(arraySize);    //size the vector
            for(int j=0; j<arraySize; j++)  //initialize elements
                hashArray[j] = NULL;
            pNonItem = new DataItem(-1);    //deleted item key is -1
        }

        void displayTable()
        {
            cout << "Table: ";
            for(int j=0; j<arraySize; j++)
            {
                if(hashArray[j] != NULL)
                    cout << hashArray[j]->iData << " ";
                else
                    cout << "** ";
            }
```

```
            cout << endl;
        }

        int hashFunc(int key)
        {
            return key % arraySize;          //hash function
        }

        void insert(DataItem* pItem)         //insert a DataItem
        {                                    //(assumes table not full)
            int key = pItem->iData;          //extract key
            int hashVal = hashFunc(key);     //hash the key
                                             //until empty cell or -1,
            while(hashArray[hashVal] != NULL && hashArray[hashVal]->iData
            != -1)
            {
                ++hashVal;                    //go to next cell
                hashVal %= arraySize;         //wraparound if necessary
            }
            hashArray[hashVal] = pItem;      //insert item
        }  //end insert()

        DataItem* remove(int key)           //remove a DataItem
        {
            int hashVal = hashFunc(key);     //hash the key

            while(hashArray[hashVal] != NULL)  //until empty cell,
            {                                  //found the key?
                if(hashArray[hashVal]->iData == key)
                {
                    DataItem* pTemp = hashArray[hashVal]; //save item
                    hashArray[hashVal] = pNonItem;        //delete item
                    return pTemp;                         //return item
                }
                ++hashVal;                    //go to next cell
                hashVal %= arraySize;         //wraparound if necessary
            }
            return NULL;                      //can't find item
        }  //end remove()

        DataItem* find(int key)              //find item with key
        {
            int hashVal = hashFunc(key);     //hash the key

            while(hashArray[hashVal] != NULL)  //until empty cell,
            {                                  //found the key?
                if(hashArray[hashVal]->iData == key)
                    return hashArray[hashVal];   //yes, return item
                ++hashVal;                    //go to next cell
                hashVal %= arraySize;         //wraparound if necessary
            }
            return NULL;                      //can't find item
        }

};  //end class HashTable
//-----------------------------------------------------------------
int main()
{
    DataItem* pDataItem;
    int aKey, size, n, keysPerCell;
    time_t aTime;
```

```
    char choice = 'b';
                                        //get sizes
    cout << "Enter size of hash table: ";
    cin >> size;
    cout << "Enter initial number of items: ";
    cin >> n;
    keysPerCell = 10;
                                        //make table
    HashTable theHashTable(size);
    srand( static_cast<unsigned>(time(&aTime)) );
    for(int j=0; j<n; j++)              //insert data
    {
        aKey = rand() % (keysPerCell*size);
        pDataItem = new DataItem(aKey);
        theHashTable.insert(pDataItem);
    }

    while(choice != 'x')                //interact with user
    {
        cout << "Enter first letter of "
             << "show, insert, delete, or find: ";
        char choice;
        cin >> choice;
        switch(choice)
        {
            case 's':
                theHashTable.displayTable();
                break;
            case 'i':
                cout << "Enter key value to insert: ";
                cin >> aKey;
                pDataItem = new DataItem(aKey);
                theHashTable.insert(pDataItem);
                break;
            case 'd':
                cout << "Enter key value to delete: ";
                cin >> aKey;
                theHashTable.remove(aKey);
                break;
            case 'f':
                cout << "Enter key value to find: ";
                cin >> aKey;
                pDataItem = theHashTable.find(aKey);
                if(pDataItem != NULL)
                    cout << "Found " << aKey << endl;
                else
                    cout << "Could not find " << aKey << endl;
                break;
            default:
                cout << "Invalid entry\n";
        }  //end switch
    }  //end while
    return 0;
}  //end main()
```

Task 1 (C++). Based on the code of Example 1 write a program implementing the hash table with a method of addressing: **quadratic probe**.

Example 2 (C++). Implementation of the hash table using **double hashing**.

[hashDouble.cpp]

```
//hashDouble.cpp
//demonstrates hash table with double hashing
#include <iostream>
#include <vector>
#include <cstdlib>                    //for random numbers
#include <ctime>                      //for random numbers
using namespace std;

class DataItem
{                                  //(could have more items)
    public:
        int iData;                        //data item (key)

        DataItem(int ii) : iData(ii)      //constructor
        {  }

};  //end class DataItem
//-----------------------------------------------------------------
class HashTable
{
    private:
        vector<DataItem*> hashArray;   //vctor holds hash table
        int arraySize;
        DataItem* pNonItem;            //for deleted items

    public:
        HashTable(int size) : arraySize(size)  //constructor
        {
            hashArray.resize(arraySize);   //size the vector
            for(int j=0; j<arraySize; j++) //initialize elements
                hashArray[j] = NULL;
            pNonItem = new DataItem(-1);
        }

        void displayTable()
        {
            cout << "Table: ";
            for(int j=0; j<arraySize; j++)
            {
                if(hashArray[j] != NULL)
                    cout << hashArray[j]->iData << " ";
                else
                    cout << "** ";
            }
            cout << endl;
        }

        int hashFunc1(int key)
        {
            return key % arraySize;
        }

        int hashFunc2(int key)
```

```
        {
            //non-zero, less than array size, different from hF1
            //array size must be relatively prime to 5, 4, 3, and 2
            return 5 - key % 5;
        }

        //insert a DataItem
        void insert(int key, DataItem* pItem)
        {                                 //(assumes table not full)
            int hashVal = hashFunc1(key);  //hash the key
            int stepSize = hashFunc2(key); //get step size
                                           //until empty cell or -1
            while(hashArray[hashVal] != NULL &&
            hashArray[hashVal]->iData != -1)
            {
                hashVal += stepSize;        //add the step
                hashVal %= arraySize;       //for wraparound
            }
            hashArray[hashVal] = pItem;    //insert item
        }  //end insert()

        DataItem* remove(int key)   //delete a DataItem
        {
            int hashVal = hashFunc1(key);  //hash the key
            int stepSize = hashFunc2(key); //get step size

            while(hashArray[hashVal] != NULL)  //until empty cell,
            {                                  //is correct hashVal?
                if(hashArray[hashVal]->iData == key)
                {
                    DataItem* pTemp = hashArray[hashVal]; //save item
                    hashArray[hashVal] = pNonItem;        //delete item
                    return pTemp;                         //return item
                }
                hashVal += stepSize;        //add the step
                hashVal %= arraySize;       //for wraparound
            }
            return NULL;                   //can't find item
        }  //end remove()

        DataItem* find(int key)           //find item with key
        {                        //(assumes table not full)
            int hashVal = hashFunc1(key);  //hash the key
            int stepSize = hashFunc2(key); //get step size

            while(hashArray[hashVal] != NULL)  //until empty cell,
            {                                  //is correct hashVal?
                if(hashArray[hashVal]->iData == key)
                    return hashArray[hashVal];   //yes, return item
                hashVal += stepSize;             //add the step
                hashVal %= arraySize;            //for wraparound
            }
            return NULL;                   //can't find item
        };

};  //end class HashTable
//-----------------------------------------------------------------
int main()
{
    int aKey;
    DataItem* pDataItem;
```

```
    int size, n;
    char choice = 'b';
    time_t aTime;
                                        //get sizes
    cout << "Enter size of hash table (use prime number): ";
    cin >> size;
    cout << "Enter initial number of items: ";
    cin >> n;
                                        //make table
    HashTable theHashTable(size);       //seed random numbers
    srand( static_cast<unsigned>(time(&aTime)) );

    for(int j=0; j<n; j++)              //insert data
    {
        aKey = rand()  % (2 * size);
        pDataItem = new DataItem(aKey);
        theHashTable.insert(aKey, pDataItem);
    }

    while(true)                         //interact with user
    {
        cout << "Enter first letter of ";
        cout << "show, insert, delete, find or quit: ";
        cin >> choice;
        switch(choice)
        {
            case 's':
                theHashTable.displayTable();
                break;
            case 'i':
                cout << "Enter key value to insert: ";
                cin >> aKey;
                pDataItem = new DataItem(aKey);
                theHashTable.insert(aKey, pDataItem);
                break;
            case 'd':
                cout << "Enter key value to delete: ";
                cin >> aKey;
                theHashTable.remove(aKey);
                break;
            case 'f':
                cout << "Enter key value to find: ";
                cin >> aKey;
                pDataItem = theHashTable.find(aKey);
                if(pDataItem != NULL)
                    cout << "Found " << aKey << endl;
                else
                    cout << "Could not find " << aKey << endl;
                break;
            case 'q':
                return 0;
            default:
                cout << "Invalid entry\n";
        }  //end switch
    }  //end while
    return 0;
}  //end main()
```

Task 2 (C++). Fill in the code given below body functions: insert(), remove(), find() and displayList(). Implementation of the hash table is intended to use a **separate chain** method (ie. *single linked list*).

[hashChain.cpp]

```
//hashChain.cpp
//demonstrates hash table with separate chaining
#include <iostream>
#include <vector>
#include <cstdlib>                    //for random numbers
#include <ctime>                      //for random numbers
using namespace std;

class Link
{                                   //(could be other items)
    public:
        int iData;                      //data item
        Link* pNext;                    //next link in list

        Link(int it) : iData(it)           //constructor
        {  }

        void displayLink()                 //display this link
        { cout << iData << " "; }

};  //end class Link
//--------------------------------------------------------------
class SortedList
{
    private:
        Link* pFirst;                   //ref to first list item

    public:
        SortedList()                       //constructor
        { pFirst = NULL; }

        void insert(Link* pLink)           //insert link, in order
        {
            int key = pLink->iData;
            Link* pPrevious = NULL;         //start at first
            Link* pCurrent = pFirst;

            /* need to be completed */

        }  //end insert()

        void remove(int key)              //delete link
        {                                 //(assumes non-empty list)
            Link* pPrevious = NULL;      //start at first
            Link* pCurrent = pFirst;

            /* need to be completed */

        }  //end remove()

        Link* find(int key)                //find link
        {
            Link* pCurrent = pFirst;        //start at first
```

```
            /* need to be completed */

        }  //end find()

        void displayList()
        {
            cout << "List (first->last): ";
            Link* pCurrent = pFirst;      //start at beginning of list

            /* need to be completed */
        }

};  //end class SortedList
//-----------------------------------------------------------------


class HashTable
{
    private:
        vector<SortedList*> hashArray;  //vector of lists
        int arraySize;

    public:
        HashTable(int size)                 //constructor
        {
            arraySize = size;
            hashArray.resize(arraySize);    //set vector size
            for(int j=0; j<arraySize; j++)  //fill vector
                hashArray[j] = new SortedList; //with lists
        }

        void displayTable()
        {
            for(int j=0; j<arraySize; j++)  //for each cell,
            {
                cout << j << ". ";            //display cell number
                hashArray[j]->displayList(); //display list
            }
        }

        int hashFunc(int key)               //hash function
        {
            return key % arraySize;
        }

        void insert(Link* pLink)            //insert a link
        {
            int key = pLink->iData;
            int hashVal = hashFunc(key);    //hash the key
            hashArray[hashVal]->insert(pLink); //insert at hashVal
        }  //end insert()

        void remove(int key)                //delete a link
        {
            int hashVal = hashFunc(key);    //hash the key
            hashArray[hashVal]->remove(key); //delete link
        }  //end remove()

        Link* find(int key)                 //find link
        {
            int hashVal = hashFunc(key);    //hash the key
```

```
            Link* pLink = hashArray[hashVal]->find(key);  //get link
            return pLink;                    //return link
        }

};  //end class HashTable
//--------------------------------------------------------------
int main()
{
    int aKey;
    Link* pDataItem;
    int size, n, keysPerCell = 100;
    time_t aTime;
    char choice = 'b';
                                    //get sizes
    cout << "Enter size of hash table: ";
    cin >> size;
    cout << "Enter initial number of items: ";
    cin >> n;
    HashTable theHashTable(size);   //make table
                                    //initialize random numbers
    srand( static_cast<unsigned>(time(&aTime)) );

    for(int j=0; j<n; j++)          //insert data
    {
        aKey = rand() % (keysPerCell * size);
        pDataItem = new Link(aKey);
        theHashTable.insert(pDataItem);
    }

    while(choice != 'x')            //interact with user
    {
        cout << "Enter first letter of ";
        cout << "show, insert, delete, find or quit: ";
        cin >> choice;
        switch(choice)
        {
            case 's':
                theHashTable.displayTable();
                break;
            case 'i':
                cout << "Enter key value to insert: ";
                cin >> aKey;
                pDataItem = new Link(aKey);
                theHashTable.insert(pDataItem);
                break;
            case 'd':
                cout << "Enter key value to delete: ";
                cin >> aKey;
                theHashTable.remove(aKey);
                break;
            case 'f':
                cout << "Enter key value to find: ";
                cin >> aKey;
                pDataItem = theHashTable.find(aKey);
                if(pDataItem != NULL)
                    cout << "Found " << aKey << endl;
                else
                    cout << "Could not find " << aKey << endl;
                break;
            case 'q':
                return 0;
```

```
            default:
                cout << "Invalid entry\n";
        }  //end switch
    }  //end while
    return 0;
}  //end main()
```

Based on:
Robert Lafore – "Teach Yourself Data Structures And Algorithms In 24 Hours"