PROJECT 1

HASHING

CSCI 230
DATA STRUCTURE II

MAI PHAM

DEVELOPMENT ENVIRONMENT
MAC OS (xCode)

TABLE OF CONTENTS

Mai Pham
CSCI 230 - Project 1
Prof. T. Vo

PROJECT STATUS

OBJECTIVE:

- Define and implement three hashing schemes (separate chaining, linear hashing, and double hashing) using the large given text file with the load factor of 0.25, 0.50, 0.75, and 0.90.

SUMMARY:

- I was really struggling and confused on what and how to do this project for days. Even when I got something to output, I was unable to tell if the result is correct or not or what the expecting result should be. Until, I finally decided to put aside the coding part and implement the hashing by hand, then I start to understand and be able continue working on my code. I think my code works correctly because the output match with the hashing I did by hand for the small text file. However, when I started to implement it on the larger file, my double hashing stopped halfway when inputting the data. I found out because the map uses bidirectional iterator which does not allow multiple increment. The advance function should solve this problem, but somehow it says no function is found. Therefore, I used a for loop instead which did solve my problem.

EXTRA CREDIT:

- I did extra credit 2; create a menu for insert, search, and remove after the initial selection. I'm not sure how much detail or what information need to provide for this extra credit. So, I just did some insertion with new key and existing key which update the value part. I also did some search which tell if the key is in the list or not and remove existing key. I also show the number of probes for each insert, search, and remove as requested, display the new list after all the work, and my program seems to work without a problem.

RESULTS & DISCOUSSIONS

SEPARATE CHAINING						
Load Factor	**Table Size**	**Average Probes**	**Max Probes**	**Number of Cluster**	**Average Cluster**	**Max Cluster**
0.25	40009	1.0781	3	761	2.01314	3
0.50	20011	1.1976	4	1578	2.12294	4
0.75	13337	1.3242	5	2242	2.20963	5
0.90	11113	1.402	5	2502	2.28098	5
LINEAR HASHING						
Load Factor	**Table Size**	**Average Probes**	**Max Probes**	**Number of Cluster**	**Average Cluster**	**Max Cluster**
0.25	40009	1.1093	9	2000	2.5815	12
0.50	20011	1.3971	18	2146	3.73719	28
0.75	13337	2.3344	92	1266	7.45182	95
0.90	11113	4.8251	271	519	18.9191	349
DOUBLE HASHING						
Load Factor	**Table Size**	**Average Probes**	**Max Probes**	**Number of Cluster**	**Average Cluster**	**Max Cluster**
0.25	40009	1.1861	9	1918	2.36861	7
0.50	20011	1.5259	14	2495	2.99639	13
0.75	13337	2.1444	20	1861	5.03009	32
0.90	11113	2.9168	56	905	10.9315	66

I think the result is reasonably correct. Based on book, separate chaining should be the most efficient hashing which can see here on the chart. As for the open addressing, the double hashing is a better choice compare to the linear hashing. Also, the chart allows us to see that as load factor decrease, the bigger the table size is, allow a smaller number of probes and cluster size.

OUTPUT FILE

```
Please wait....
Reading data from file...

Menu
1. Display Hashing
2. Insert new key/value
3. Search for key/value
4. Remove a key/value
5. Display the list
6. Quit

Please choose an option ==> 1
            SEPARATE HASHING
=======================================
Load Factor 0.25
Table size: 40009
Total Probes: 10781
Average number of probes: 1.0781
Maximum number of probes for worst case: 3
Number of clusters: 761
Average clusters size: 2.01314
Largest cluster size: 3

Load Factor 0.50
Table size: 20011
Total Probes: 11976
Average number of probes: 1.1976
Maximum number of probes for worst case: 4
Number of clusters: 1578
Average clusters size: 2.12294
Largest cluster size: 4

Load Factor 0.75
Table size: 13337
Total Probes: 13242
Average number of probes: 1.3242
Maximum number of probes for worst case: 5
Number of clusters: 2242
Average clusters size: 2.20963
Largest cluster size: 5

Load Factor 0.90
Table size: 11113
Total Probes: 14020
Average number of probes: 1.402
Maximum number of probes for worst case: 5
Number of clusters: 2502
```

```
Average clusters size: 2.28098
Largest cluster size: 5

            LINEAR HASHING
=======================================
Load Factor 0.25
Table size: 40009
Total Probes: 11093
Average number of probes: 1.1093
Maximum number of probes for worst case: 9
Number of clusters: 2000
Average clusters size: 2.5815
Largest cluster size: 12

Load Factor 0.50
Table size: 20011
Total Probes: 13971
Average number of probes: 1.3971
Maximum number of probes for worst case: 18
Number of clusters: 2146
Average clusters size: 3.73719
Largest cluster size: 28

Load Factor 0.75
Table size: 13337
Total Probes: 23344
Average number of probes: 2.3344
Maximum number of probes for worst case: 92
Number of clusters: 1266
Average clusters size: 7.45182
Largest cluster size: 95

Load Factor 0.90
Table size: 11113
Total Probes: 48251
Average number of probes: 4.8251
Maximum number of probes for worst case: 271
Number of clusters: 519
Average clusters size: 18.9191
Largest cluster size: 349

            DOUBLING HASHING
=======================================
Load Factor 0.25
Table size: 40009
Total Probes: 11861
Average number of probes: 1.1861
Maximum number of probes for worst case: 9
Number of clusters: 1918
Average clusters size: 2.36861
Largest cluster size: 7

Load Factor 0.50
Table size: 20011
Total Probes: 15259
```

```
Average number of probes: 1.5259
Maximum number of probes for worst case: 14
Number of clusters: 2495
Average clusters size: 2.99639
Largest cluster size: 13

Load Factor 0.75
Table size: 13337
Total Probes: 21444
Average number of probes: 2.1444
Maximum number of probes for worst case: 20
Number of clusters: 1861
Average clusters size: 5.03009
Largest cluster size: 32

Load Factor 0.90
Table size: 11113
Total Probes: 29168
Average number of probes: 2.9168
Maximum number of probes for worst case: 56
Number of clusters: 905
Average clusters size: 10.9315
Largest cluster size: 66

Menu
1. Display Hashing
2. Insert new key/value
3. Search for key/value
4. Remove a key/value
5. Display the list
6. Quit

Please choose an option ==> 6
Thank you for using my program.
Program ended with exit code: 0
```

EXTRA CREDIT

```
================================================================================
Please wait....
Reading data from file...

Menu
1. Display Hashing
2. Insert new key/value
3. Search for key/value
4. Remove a key/value
5. Display the list
6. Quit

Please choose an option ==> 2
Please enter the key: 13
Please enter the value: hello
It cost 1 probe(s) for this work
Insert completed.
```

```
Menu
1. Display Hashing
2. Insert new key/value
3. Search for key/value
4. Remove a key/value
5. Display the list
6. Quit

Please choose an option ==> 2
Please enter the key: 28
Please enter the value: world
It cost 2 probe(s) for this work
Insert completed.

Menu
1. Display Hashing
2. Insert new key/value
3. Search for key/value
4. Remove a key/value
5. Display the list
6. Quit

Please choose an option ==> 3
Please enter the search key: 37
It cost 1 probe(s) for this work
37 is in the list with value 73

Menu
1. Display Hashing
2. Insert new key/value
3. Search for key/value
4. Remove a key/value
5. Display the list
6. Quit

Please choose an option ==> 3
Please enter the search key: 22
It cost 1 probe(s) for this work
22 is not in the list

Menu
1. Display Hashing
2. Insert new key/value
3. Search for key/value
4. Remove a key/value
5. Display the list
6. Quit

Please choose an option ==> 4
Please enter the key to remove: 5
It cost 1 probe(s) for this work
Remove completed.

Menu
1. Display Hashing
```

```
2. Insert new key/value
3. Search for key/value
4. Remove a key/value
5. Display the list
6. Quit

Please choose an option ==> 5
Print all entries (chain map)
(28,world)
(13,hello)
(37,73)
(15,51)
(21,12)

Menu
1. Display Hashing
2. Insert new key/value
3. Search for key/value
4. Remove a key/value
5. Display the list
6. Quit

Please choose an option ==> 6
Thank you for using my program.
Program ended with exit code: 0
```

SOURCE CODE

Main.cpp
Note: Does not include Entry.h

```
//
//  main.cpp
//  Project 1
//
//  Created by Mai Pham on 3/14/18.
//  Copyright © 2018 Mai Pham. All rights reserved.
//

#include "ChainHashing.h"
#include "LinearHashing.h"
#include "DoubleHashing.h"
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

int tableSize(int num, double lf);
bool isPrime(int n);

int main()
{
    int size, key, menu;
    string value;
    int lf25, lf50, lf75, lf90;
```

```
ifstream myReadList;
myReadList.open("p1large.txt");
if(!myReadList.is_open())
    cout << "No text file found. " << endl;

// read in # of entries
myReadList >> size;
// get prime # for table size base on load factor
lf25 = tableSize(size, 0.25);
lf50 = tableSize(size, 0.50);
lf75 = tableSize(size, 0.75);
lf90 = tableSize(size, 0.90);

// a (int, string) map with (table size)
HashMap<int, string> ChainMap1(lf25);
HashMap<int, string> ChainMap2(lf50);
HashMap<int, string> ChainMap3(lf75);
HashMap<int, string> ChainMap4(lf90);

LHashMap<int, string> LinearMap1(lf25);
LHashMap<int, string> LinearMap2(lf50);
LHashMap<int, string> LinearMap3(lf75);
LHashMap<int, string> LinearMap4(lf90);

DHashMap<int, string> DoubleMap1(lf25);
DHashMap<int, string> DoubleMap2(lf50);
DHashMap<int, string> DoubleMap3(lf75);
DHashMap<int, string> DoubleMap4(lf90);

cout << "Please wait...." << endl;
cout << "Reading data from file..." << endl << endl;
while (myReadList >> key)
{
    // read in the key
    // myReadList >> key;
    // convert key to string and reverse it
    value = to_string(key);
    reverse(value.begin(), value.end());

    ChainMap1.put(key, value);
    ChainMap2.put(key, value);
    ChainMap3.put(key, value);
    ChainMap4.put(key, value);

    LinearMap1.put(key, value);
    LinearMap2.put(key, value);
    LinearMap3.put(key, value);
    LinearMap4.put(key, value);

    DoubleMap1.put(key, value);
    DoubleMap2.put(key, value);
    DoubleMap3.put(key, value);
    DoubleMap4.put(key, value);
}
```

```
cout << "Menu" << endl;
cout << "1. Display Hashing" << endl;
cout << "2. Insert new key/value" << endl;
cout << "3. Search for key/value" << endl;
cout << "4. Remove a key/value" << endl;
cout << "5. Display the list" << endl;
cout << "6. Quit" << endl;
cout << "\nPlease choose an option ==> ";
cin >> menu;

while (menu > 0 && menu < 6){
    switch (menu) {
        case 1:  {
            cout << "            SEPARATE HASHING" << endl;
            cout << "=======================================" << endl;
            cout << "Load Factor 0.25 " << endl;
            ChainMap1.printData();
            cout << "Load Factor 0.50 " << endl;
            ChainMap2.printData();
            cout << "Load Factor 0.75 " << endl;
            ChainMap3.printData();
            cout << "Load Factor 0.90 " << endl;
            ChainMap4.printData();

            cout << "            LINEAR HASHING" << endl;
            cout << "=======================================" << endl;
            cout << "Load Factor 0.25 " << endl;
            LinearMap1.printData();
            cout << "Load Factor 0.50 " << endl;
            LinearMap2.printData();
            cout << "Load Factor 0.75 " << endl;
            LinearMap3.printData();
            cout << "Load Factor 0.90 " << endl;
            LinearMap4.printData();

            cout << "            DOUBLING HASHING" << endl;
            cout << "=======================================" << endl;
            cout << "Load Factor 0.25 " << endl;
            DoubleMap1.printData();
            cout << "Load Factor 0.50 " << endl;
            DoubleMap2.printData();
            cout << "Load Factor 0.75 " << endl;
            DoubleMap3.printData();
            cout << "Load Factor 0.90 " << endl;
            DoubleMap4.printData();
            break;
        }
        case 2: {
            cout << "Please enter the key: ";
            cin >> key;
            cout << "Please enter the value: ";
            cin >> value;
            ChainMap1.put(key, value);
            ChainMap1.printProbes();
```

```
                cout << "Insert completed." << endl;
                break;
            }
            case 3: {
                cout << "Please enter the search key: ";
                cin >> key;
                HashMap<int, string>::Iterator s1 = ChainMap1.begin();
                s1 = ChainMap1.find(key);
                ChainMap1.printProbes();
                if (s1 == ChainMap1.end())
                    cout << key << " is not in the list" << endl;
                else
                    cout << key << " is in the list with value " <<
(*s1).value() << endl;
                break;
            }
            case 4: {
                cout << "Please enter the key to remove: ";
                cin >> key;
                ChainMap1.erase(key);
                ChainMap1.printProbes();
                cout << "Remove completed." << endl;
                break;
            }
            case 5: {
                cout << "Print all entries (chain map)\n";
                HashMap<int, string>::Iterator c1 = ChainMap1.begin();
                for (c1; !(c1 == ChainMap1.end()); ++c1) {
// print all entries
                    cout << "(" << (*c1).key() << "," << (*c1).value() <<
")\n";
                }
                /*
                cout << "RePrint all entries, linear map\n";
                for (l1 = LinearMap1.begin(); !(l1 == LinearMap1.end());
++l1) {  // print all entries
                    cout << "(" << (*l1).key() << "," << (*l1).value() <<
")\n";
                }*/
                break;
            }
            default:
                break;
        }
        cout << "\nMenu" << endl;
        cout << "1. Display Hashing" << endl;
        cout << "2. Insert new key/value" << endl;
        cout << "3. Search for key/value" << endl;
        cout << "4. Remove a key/value" << endl;
        cout << "5. Display the list" << endl;
        cout << "6. Quit" << endl;
        cout << "\nPlease choose an option ==> ";
        cin >> menu;
    }
    cout << "Thank you for using my program." << endl;
```

```
    return 0;
}

int tableSize(int num, double lf)   {
    int prime = num/lf+0.5;
    while (isPrime(prime) == false)
        prime++;
    return prime;
}

bool isPrime(int num)   {
    if (num <= 3)
        return num > 1;
    else if (num % 2 == 0 || num % 3 == 0)
        return false;
    else {
        for (int i = 5; i * i <= num; i += 6)
            if (num % i == 0 || num % (i + 2) == 0)
                return false;
        return true;
    }
}
```

ChainHashing.h
Note: Only include modified functions/information

```
public:
     void cluster();
    void printData();
    void printProbes();
.
.
.
private:
    int n;                                          // number of entries
    //H hash;                                       // the hash comparator
    BktArray B;                                     // bucket array
    int probes = 0;
    int totalProbes = 0;
    int maxProbes = 0;
    int numCluster = 0;
    int maxCluster = 0;
    int totalCluster = 0;
.
.
.
template <typename K, typename V>                   // find utility
typename HashMap<K,V>::Iterator HashMap<K,V>::finder(const K& k) {
    probes = 0;
    int i = k % B.size();                           // get hash index i
    BItor bkt = B.begin() + i;                      // the ith bucket
    Iterator p(B, bkt, bkt->begin());               // start of ith bucket
    probes++;
    while (!endOfBkt(p) && (*p).key() != k)         // search for k
    {
        nextEntry(p);
```

```
        probes++;
    }
    totalProbes += probes;
    if (probes > maxProbes)
        maxProbes = probes;
    return p;                                       // return final position
}

template <typename K, typename V>
void HashMap<K,V>::cluster()    {
    for (BItor bkt = B.begin(); bkt != B.end(); ++bkt) {
        if ((*bkt).size() > 1)  {
            numCluster++;
            totalCluster += (*bkt).size();
            if ((*bkt).size() > maxCluster)
                maxCluster = (*bkt).size();
        }
    }
}

template <typename K, typename V>
void HashMap<K,V>::printData()    {
    cluster();
    cout << "Table size: " << B.size() << endl;
    cout << "Total Probes: " << totalProbes << endl;
    cout << "Average number of probes: " << (double)(totalProbes) / n <<
endl;
    cout << "Maximum number of probes for worst case: " << maxProbes << endl;
    cout << "Number of clusters: " << numCluster << endl;
    if (numCluster == 0)
        cout << "Average clusters size: " << numCluster << endl;
    else
        cout << "Average clusters size: " <<
(double)(totalCluster)/numCluster << endl;
    cout << "Largest cluster size: " << maxCluster << endl << endl;
}

template <typename K, typename V>
void HashMap<K,V>::printProbes()    {
    cout << "It cost " << probes << " probe(s) for this work" << endl;
}
```

LinearHashing.h
Note: Only include functions different from ChainHashing.h

```
template <typename K, typename V>                   // find utility
typename LHashMap<K,V>::Iterator LHashMap<K,V>::finder(const K& k) {
    int probes = 0;
    int i = k % B.size();                           // get hash index i
    BItor bkt = B.begin() + i;                      // the ith bucket
    Iterator p(B, bkt, bkt->begin());               // start of ith bucket
    probes++;
    while (!endOfBkt(p) && (*p).key() != k)         // search for k
    //while (!bkt->empty())
    {
        bkt++;
```

```
        if (bkt == B.end())
            bkt = B.begin();
        //else
        //    bkt++;
        //nextEntry(p);
        p = Iterator (B, bkt, bkt->begin());
        probes++;
    }
    totalProbes += probes;
    if (probes > maxProbes)
        maxProbes = probes;
    return p;                                          // return final position
}

template <typename K, typename V>
void LHashMap<K,V>::cluster()    {
    int cluster = 0, cluster2 = 0;
    BItor bkt = B.begin();
    while (bkt != B.end()) {
        if (!(bkt->empty()))
            cluster++;
        else    {
            if (cluster > 1)    {
                numCluster++;
                totalCluster += cluster;
                if (cluster > maxCluster)
                    maxCluster = cluster;
            }
            cluster = 0;
        }
        bkt++;
    }
    if (bkt == B.end() && !(bkt->empty())) {
        if (cluster > 1)    {
            numCluster++;
            totalCluster += cluster;
            if (cluster > maxCluster)
                maxCluster = cluster;
        }
        BItor bkt1 = B.begin();
        if (!(bkt1->empty()))   {
            while (!(bkt1->empty()))    {
                cluster2++;
                bkt1++;
            }
            cluster += cluster2;
            if (cluster > maxCluster)
                maxCluster = cluster;
            numCluster--;
        }
    }
}
```

DoubleHashing.h
Note: Only include functions different from LinearHashing.h

```
template <typename K, typename V>                          // find utility
typename DHashMap<K,V>::Iterator DHashMap<K,V>::finder(const K& k) {
    int probes = 0;
    int temp = 0;
    int i = k % B.size();                                  // get hash index i
    BItor bkt = B.begin() + i;                             // the ith bucket
    Iterator p(B, bkt, bkt->begin());                      // start of ith bucket
    probes++;
    while (!endOfBkt(p) && (*p).key() != k)
    {
        // bkt++;
        temp = 7919 - (k % 7919);
        //advance(bkt, temp);
        for (int i = 0; i < temp; i++)
        {
            bkt++;
            if (bkt == B.end())
            {
                bkt = B.begin();
                i++;
            }
        }
        //advance(bkt, B.begin() + i % B.size());
        //bkt = B.begin() + ((i + temp) % B.size());
        //nextEntry(p);
        p = Iterator (B, bkt, bkt->begin());
        probes++;
    }
    totalProbes += probes;
    if (probes > maxProbes)
        maxProbes = probes;
    return p;                                          // return final position
}
```