```cpp
//***************************************************************
// Program #1 Hashing Experiment
// Name: Ben Diekhoff
// CMPS 5243 Algorithms
// Dr. Halverson
// Date: 03/24/2020
//***************************************************************
/*
This program takes a set of keys, hashes them using linear probing and double
hashing, and returns the complete hash tables for each CRP as well as the
average number of probes for each method.
/******************************************************************/
#include <iostream>
#include <iomanip>
#include <fstream>
#include <utility>
#include <ctime>
using namespace std;
ofstream outfile("output.txt");
ifstream datafile("datafile.txt");



class HASH {
private:
    pair<int, int>* table;                      // the hash table
    void Clean_Table(int);
    int Lin_Probe(int, int, int);
    int Double_Probe(int, int, int);
    void Print_Table(int);
public:
    HASH(int);                                // constructor
    ~HASH();                                  // destructor
    int Mod_Hash(int, int);
    void Clean_Table_Pub(int);                // caller function
    int Lin_Probe_Pub(int, int, int);         // caller function
    int Double_Probe_Pub(int, int, int);      // caller function
    void Print_Table_Pub(int);                // caller function

};


/**********************************************************************
    Clean_Table
    Parameters: One int

    Sets every value in the dyanimcally allocated table to a sentinel value.
***********************************************************************/
void HASH::Clean_Table(int table_size) {
    for (int i = 0; i < table_size; i++) {
        table[i].first = -9999; //table[i].first holds keys
        table[i].second = 0;   //table[i].second holds a probe count
    }
}


/*********************************************************************
    Lin_Probe
    Parameters: Three ints
    Inserts a key into an "empty" slot in the table (orig_loc by default),
    where -9999 is considered empty. If the orig_loc isn't empty, it increments
    through the table one index at a time (mod table_size) until an empty slot
    is found. It returns the number of probes it takes for successful
    insertion, as well as storing that value in table[oirg_loc].second.
*********************************************************************/
int HASH::Lin_Probe(int key, int orig_loc, int table_size) {
    int count = 1;

    //table[i].first holds keys
    //table[i].second holds a probe count
    while (table[orig_loc].first != -9999) {
        count++;
        orig_loc = (orig_loc + 1) % table_size;
    }
    table[orig_loc].first = key;
    table[orig_loc].second = count;
    return count;
}



/*********************************************************************
    Double_Probe
    Parameters: Three ints
    Inserts a key into an "empty" slot in the table (orig_loc by default),
    where -9999 is considered empty. If the orig_loc isn't empty, it increments
    through the table based on the last digit of the key + 1. When it finds an
    empty slot the key is inserted. It returns the number of probes it takes
    for successful insertion, as well as storing that value in
    table[oirg_loc].second.
*********************************************************************/
int HASH::Double_Probe(int key, int orig_loc, int table_size) {
    int increment = (key % 10) + 1;
    int count = 1;

    //table[i].first holds keys
    //table[i].second holds a probe count
    while (table[orig_loc].first != -9999) {
        count++;
        orig_loc = (orig_loc + increment) % table_size;
    }
    table[orig_loc].first = key;
    table[orig_loc].second = count;
    return count;
```

```
}


/**********************************************************************
    Print_Table
    Parameters: One int
    Prints the table in a nice, readable format to a .txt file.
    -9999 as a value under KEY means there was nothing inserted into
    that location. The same is true whe PROBES is 0.
***********************************************************************/
void HASH::Print_Table(int table_size) {
    outfile << setw(18) << left << "LOCATION" << setw(10) << "KEY"
        << setw(10) << "PROBES" << "\n";

    outfile << "\n";

    //table[i].first holds keys
    //table[i].second holds a probe count
    for (int loc = 0; loc < table_size; loc++) {
        outfile << left << setw(18) << loc << setw(10) << table[loc].first
            << setw(10) << table[loc].second << "\n";
    }
}


//////////////////////////////////////////////////////////////////////////
////////////////////////////                  ////////////////////////////
//////////////////////////// PUBLIC FUCNTIONS ////////////////////////////
////////////////////////////                  ////////////////////////////
//////////////////////////////////////////////////////////////////////////


/**********************************************************************
    HASH
    Parameters: One int
    Constructor for the HASH class.
***********************************************************************/
HASH::HASH(int table_size) {
    table = new pair<int, int>[table_size];
    //table[i].first holds keys
    //table[i].second holds a probe count
}


/**********************************************************************
    HASH
    Parameters: None
    Destructor for the HASH class.
***********************************************************************/
HASH::~HASH() {
    delete[] table;
    table = nullptr;
```

```cpp
}


/**********************************************************************
    Mod_Hash
    Parameters: Two ints
    Calculates and returns the initial hash location for a given key.
***********************************************************************/
int HASH::Mod_Hash(int key, int table_size) {
    return (key % table_size);
}


/**********************************************************************
    Clean_Table_Pub
    Parameters: Three ints
    Caller function for Clean_Table, which:
    Sets every value in the dyanimcally allocated table to -9999.
***********************************************************************/
void HASH::Clean_Table_Pub(int table_size) {
    Clean_Table(table_size);
}


/**********************************************************************
    Lin_Probe_Pub
    Parameters: Three ints
    Caller function for Lin_Probe, which:
    Inserts a key into an "empty" slot in the table (orig_loc by default),
    where -9999 is considered empty. If the orig_loc isn't empty, it increments
    through the table one index at a time (mod table_size) until an empty slot
    is found. It returns the number of probes it takes for successful
    insertion, as well as storing that value in table[oirg_loc].second.
***********************************************************************/
int HASH::Lin_Probe_Pub(int key, int orig_loc, int table_size) {
    return Lin_Probe(key, orig_loc, table_size);
}


/**********************************************************************
    Double_Probe_Pub
    Parameters: Three ints
    Caller function for Double_Probe, which:
    Inserts a key into an "empty" slot in the table (orig_loc by default),
    where -9999 is considered empty. If the orig_loc isn't empty, it increments
    through the table based on the last digit of the key + 1. When it finds an
    empty slot the key is inserted. It returns the number of probes it takes
    for successful insertion, as well as storing that value in
    table[oirg_loc].second.
***********************************************************************/
int HASH::Double_Probe_Pub(int key, int orig_loc, int table_size) {
    return Double_Probe(key, orig_loc, table_size);
```

```cpp
}


/**********************************************************************
    Print_Table_Pub
    Parameters: One int
    Caller function for Print_Table, which:
    Prints the table in a nice, readable format to a .txt file.
    -9999 as a value under KEY means there was nothing inserted into
    that location. The same is true whe PROBES is 0.
***********************************************************************/
void HASH::Print_Table_Pub(int table_size) {
    Print_Table(table_size);
}

//////////////////////////////////////////////////////////////////////////
////////////////////////////                  ////////////////////////////
////////////////////////////   MAIN FUCNTION  ////////////////////////////
////////////////////////////                  ////////////////////////////
//////////////////////////////////////////////////////////////////////////

int main() {

    const int table_size = 311;
    const int randData_size = 250;
    int probe_count = 0;  // 0 is a sentinel value
    int orig_loc = -1;  // Location a key is initially hashed toS
    double avg_probes = -1.1; // Avg # of probes for inserting into a table
    int randData[randData_size] = { 0 }; // holds random numbers to be hashed
    double n;  // number of items to insert
    double alpha;  // Load factor
    int key;        //The value to be hashed
    HASH h(table_size);

    // Run all experiments twice
        // Load the first half of the dataset into randData on iteration 0
        // Load the second half on iteration 1
    for (int iter = 0; iter < 2; iter++) {
                for (int i = 0; i < randData_size; i++){
                    datafile >> randData[i];
                }
        // Loop through the 4 experiments
        for (int i = 0; i < 4; i++) {

            //make sure variables are reset
            probe_count = 0;
            orig_loc = -1;
            avg_probes = -1.1;
            probe_count = 0;
            h.Clean_Table_Pub(table_size); // Set all values in h.table to -9999

            //determine how much data to insert into the hash table
            if (i == 0 || i == 2)
                n = 205;
            else
                n = 250;

            alpha = n / table_size;

            // Linear Probe
            if (i < 2) {
                for (int i = 0; i < n; i++) {
                    key = randData[i];
                    // determines hash location for key
                    orig_loc = h.Mod_Hash(key, table_size);
                    // accumulator for the number of probes required to insert
                    // all values into the table
                    probe_count += h.Lin_Probe_Pub(key, orig_loc, table_size);
                }
                outfile << "LINEAR PROBE WITH ALPHA = " << alpha
                    << "\nIteration: " << iter + 1 << "\n==============\n";

            }

            // Double Hash
            else {
                for (int i = 0; i < n; i++) {
                    key = randData[i];
                    // determines hash location for key
                    orig_loc = h.Mod_Hash(key, table_size);
                    // accumulator for the number of probes required to insert
                    // all values into the table
                    probe_count += h.Double_Probe_Pub(key, orig_loc, table_size);
                }
                outfile << "DOUBLE HASH WITH ALPHA " << alpha
                    << "\nIteration: " << iter + 1 << "\n==============\n";
            }

            // Print information to outfile
            avg_probes = probe_count / n;
            h.Print_Table_Pub(table_size);
            outfile << "\nAvg # of probes: " << fixed << setprecision(3)
                << setw(21) << avg_probes << "\n\n";
        }
    }
    outfile.close();
    datafile.close();
    return 0;
}
```