

**LAB5.h**

#include <iostream>

#include <random>

#include <string>

using namespace std;

struct Key {

char value[6];

Key() {

for (int i = 0; i < 6; i++) value[i] = 0;

}

};

struct Keys {

Key\* key;

int length;

Keys(int length) {

this->length = length;

key = new Key[length];

for (int i = 0; i < length; i++) key[i] = Key();

}

~Keys() {

delete[] key;

}

};

class Table {

private:

int max\_column\_size;

public:

int display\_w\_limit = 168;

int display\_h = 8;

int length;

int\* inclusions;

void put(int index) {

if(inclusions[index] > max\_column\_size)

max\_column\_size = inclusions[index];

inclusions[index]++;

}

void clear() {

for (int i = 0; i < length; i++) inclusions[i] = 0;

max\_column\_size = 0;

}

Table(int length) {

this->length = length;

inclusions = new int[length];

clear();

}

~Table(){

delete[] inclusions;

}

operator string() {

string result = "";

int empty\_cells = 0;

int single\_cells = 0;

int collision\_cells = 0;

int collision\_key\_sum = 0;

float avg\_collisions = 0;

for (int i = 0; i < length; i++) {

avg\_collisions += max(0, inclusions[i] - 1);

collision\_key\_sum += (inclusions[i] > 1) ? inclusions[i] : 0;

if (inclusions[i] == 0) empty\_cells++;

else if (inclusions[i] == 1) single\_cells++;

else collision\_cells++;

}

avg\_collisions /= collision\_cells;

result += " ";

for (int i = 0; i < min(length, display\_w\_limit); i++) result += '-';

result += '\n';

for (int row = 0; row < display\_h; row++) {

result += '|';

int step = 0;

int empty\_count = 0;

float acc = 0;

for (int i = 0; i < length; i++) {

if (inclusions[i] - 1 <= 0) empty\_count++;

acc = max((float)inclusions[i] - 1, acc);

if (i < step \* (length / (float)display\_w\_limit)) continue;

if (acc == 0) acc = -1;

result +=

(acc >= (display\_h - row) \* (max\_column\_size / (float)display\_h)) ? (empty\_count == 0) ? (char)178 : (empty\_count == 1) ? (char)177 : (char) 176 :

(acc && acc >= (display\_h - row - 1) \* (max\_column\_size / (float)display\_h)) ? (empty\_count == 0) ? (char)177 : (empty\_count == 1) ? (char)176 : '\_'

: ' ';

acc = 0;

empty\_count = 0;

step++;

}

string white = "\033[107m\033[30m\033[4m";

string black = "\033[49m\033[0m\033[24m";

if (row == 0) result += "[0] EMPTY: " + to\_string(empty\_cells) + " CELLS";

else if (row == 1) result += "[1] SINGLE: " + white + to\_string(single\_cells) + black + " CELLS (KEYS)";

else if (row == 2) result += "[N] COLLISION: " + to\_string(collision\_cells) + " CELLS (" + white + to\_string(collision\_key\_sum) + black + " KEYS)";

//else if (row == 3) result += "[N~] AVG COLLISION: " + to\_string(avg\_collisions) + " KEYS";

else if (row == 3) result += "[Nm] MAX COLLISION: " + to\_string(max\_column\_size+1) + " KEYS";

result += "\n";

}

result += " ";

for (int i = 0; i < min(length, display\_w\_limit); i++) result += '-';

result += '\n';

return result;

}

};

int hash\_div(int key, int table\_size);

int hash\_mul(int key, int table\_size);

int hash\_add(Key key); //288 <= hash <= 732 (445)

int hash\_xor(Key key);

int get\_rand();

**hash.cpp**

#include "LAB5.h"

int get\_rand() {

char c[4] = { rand() % 256, rand() % 256, rand() % 256, rand() % 128 };

int result = (unsigned)\*((int\*)c);

return result;

}

int hash\_div(int key, int table\_size) {

return key % table\_size;

}

int hash\_mul(int key, int table\_size) {

double rand\_value = (sqrt(5) - 1) \* 0.5;//0,61803398875...

double m = rand\_value \* key;

return table\_size \* (m - floor(m));

}

int hash\_add(Key key) {

int hash = 0;

for (int i = 0; i < 6; i++) {

hash += key.value[i];

}

//return get\_rand();

return hash;

}

//int ii = 0;

char xor\_array[6] = { rand() % 256, rand() % 256, rand() % 256, rand() % 256, rand() % 256, rand() % 256};

int hash\_xor(Key key) {

int hash = 0;

for (int i = 0; i < 6; i++) {

hash += 127 & ((key.value[i] ^ xor\_array[i % 6]));

}

//ii += 2;

//return (int)(ii);

//return get\_rand();

return hash;

}

**AISD\_LAB5.cpp**

#include "LAB5.h"

void fill(Keys& keys) {

for (int i = 0; i < keys.length; i++) {

Key& key = keys.key[i];

for (int j = 0; j < 6; j++) {

char& c = key.value[j];

int r = rand()%62;

c = r;

c +=

(0 <= r && r <= 9) ? '0' - 0 :

(10 <= r && r <= 35) ? 'A' - 10 :

(36 <= r && r <= 61) ? 'a' - 36 :

0;

}

}

}

int main() {

cout << "\033[1m";

int table\_size, key\_size;

key\_size = 400;

table\_size = 400;

cout << "key\_size(int) table\_size(int) >> ";

cin >> key\_size >> table\_size;

system("cls");

cout << key\_size << " keys on " << table\_size << " addresses:\n";

//while (true)

{

Table tableMA = Table(table\_size);

Table tableDA = Table(table\_size);

Table tableMX = Table(table\_size);

Table tableDX = Table(table\_size);

Keys keys = Keys(key\_size);

fill(keys);

int ha = 0, hx = 0;

for (int i = 0; i < keys.length; i++) {

int hash;

int ha = hash\_add(keys.key[i]);

int hx = hash\_xor(keys.key[i]);

hash = hash\_mul(ha, tableMA.length);

tableMA.put(hash);

hash = hash\_div(ha, tableDA.length);

tableDA.put(hash);

hash = hash\_mul(hx, tableMX.length);

tableMX.put(hash);

hash = hash\_div(hx, tableDX.length);

tableDX.put(hash);

//cout << keys.key[i].value[0] << keys.key[i].value[1] << keys.key[i].value[2] << keys.key[i].value[3] << keys.key[i].value[4] << keys.key[i].value[5] << endl;

}

for (int i = 0; i < min(tableMA.length, tableMA.display\_w\_limit) / 2 - 5; i++) cout << ' '; cout << "MUL & ADD:\n" << (string)tableMA << endl;

for (int i = 0; i < min(tableMA.length, tableDA.display\_w\_limit) / 2 - 5; i++) cout << ' '; cout << "DIV & ADD:\n" << (string)tableDA << endl;

for (int i = 0; i < min(tableMA.length, tableMX.display\_w\_limit) / 2 - 5; i++) cout << ' '; cout << "MUL & XOR:\n" << (string)tableMX << endl;

for (int i = 0; i < min(tableMA.length, tableDX.display\_w\_limit) / 2 - 5; i++) cout << ' '; cout << "DIV & XOR:\n" << (string)tableDX << endl;

system("pause");

}

return 0;

}