Aaron Williams

Assignment 4 (Homework Unit 5)

15 September 2017

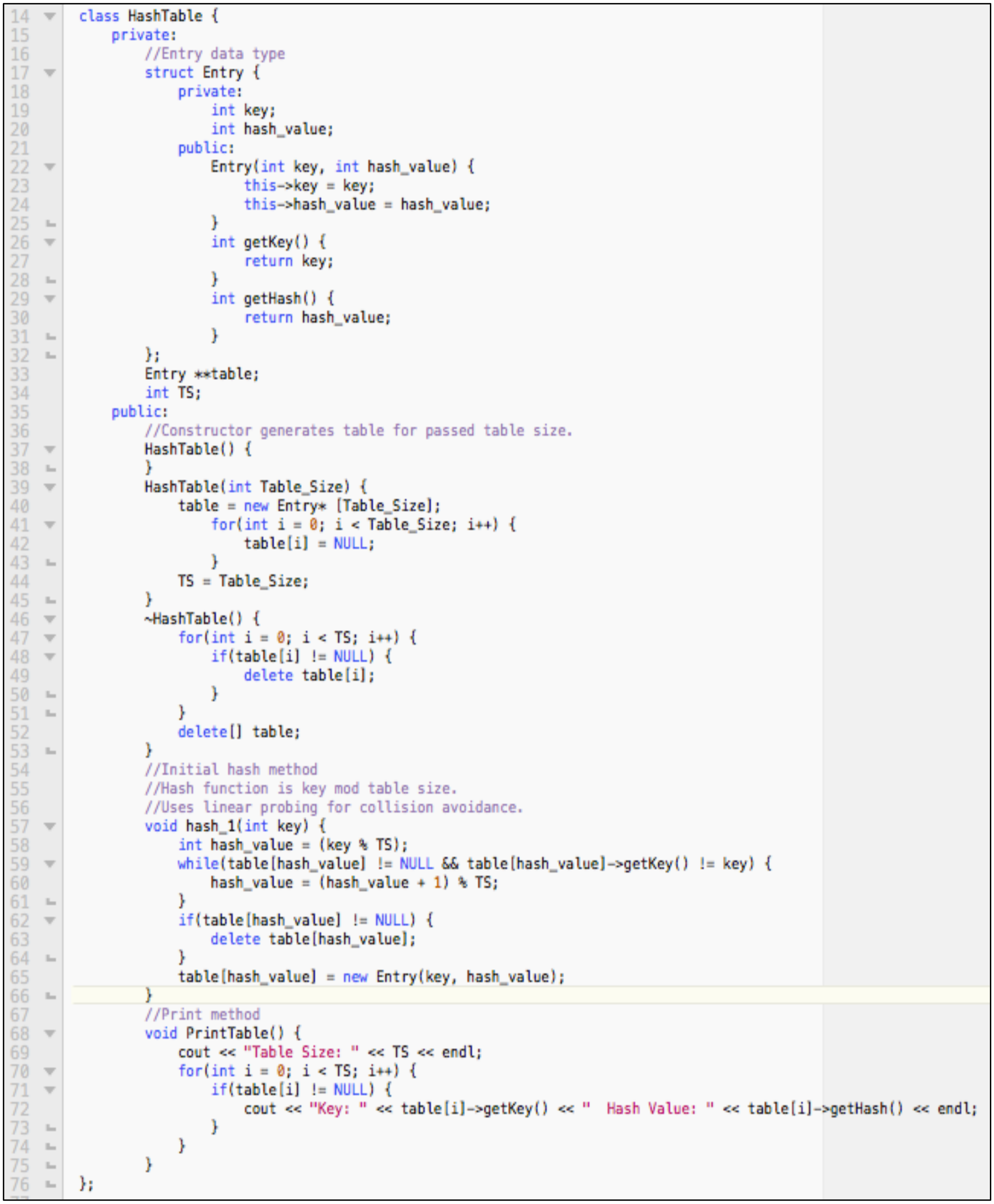
Assignment Description

The purpose of this assignment is to develop a hash.cpp program that inserts the sequence of integers {121, 81, 16, 100, 25, 0, 1, 9, 4, 36, 64, 49} into hash tables based on different table sizes and hash functions used. Part 1 uses a table size of 17. Part 2 implements rehashing and uses an initial table size of 7.

Logic and Outputs

1) For part 1, the goal is to take the 12-integer sequence and generate an adequate hash table for a table size of 17 while avoiding collisions. The first step was to develop a HashTable class that could construct different sized hash tables or use different hash functions. The HashTable class was created using the struct Entry. The purpose of creating this new data type is to store two different paramaters, key and hash value, for each occurrence in the hash table. The class constructor takes in a parameter for table size and constructs a table of that many entries. The destructor, initial hash, and print methods were also created for the class. The initial hash method takes in a key parameter and hashes the key using the function 'key mod table size'. In addition, the collision avoidance strategy of linear probing was used for the initial hash method. Once the key is hashed, the entry containing the key and associated hash value is inserted into the next open position in the hash table. The print method first prints the table size for the HashTable, and then prints each key alongside its hash value.

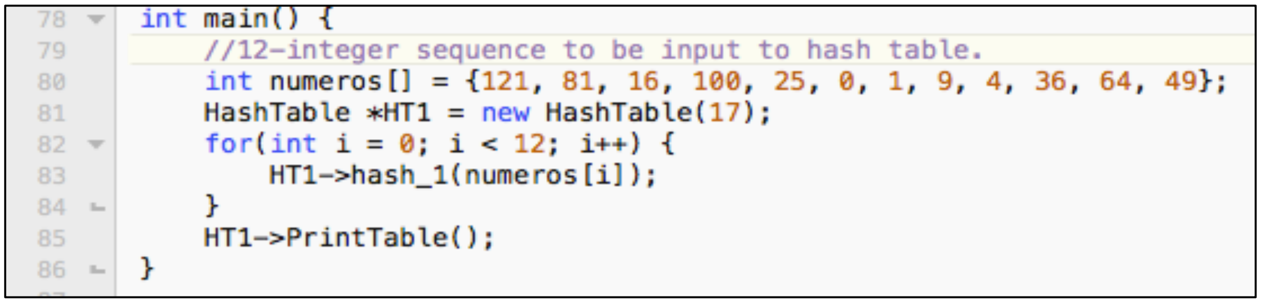
HashTable Class:



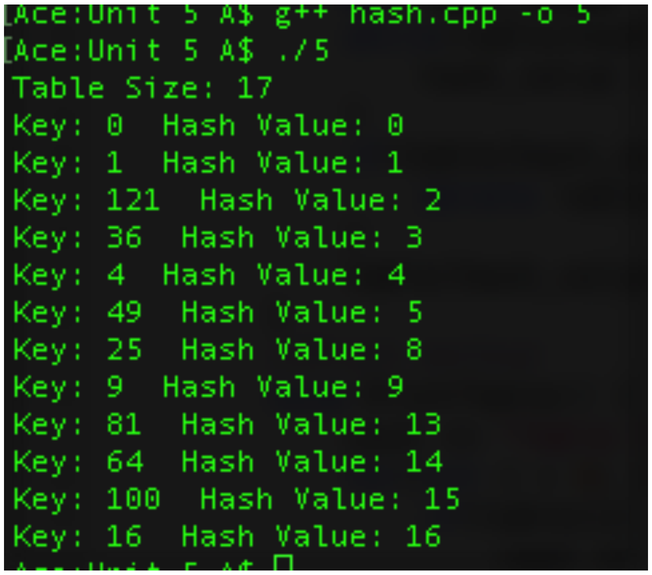
```cpp
class HashTable {
    private:
        //Entry data type
        struct Entry {
            private:
                int key;
                int hash_value;
            public:
                Entry(int key, int hash_value) {
                    this->key = key;
                    this->hash_value = hash_value;
                }
                int getKey() {
                    return key;
                }
                int getHash() {
                    return hash_value;
                }
        };
        Entry **table;
        int TS;
    public:
        //Constructor generates table for passed table size.
        HashTable() {
        }
        HashTable(int Table_Size) {
            table = new Entry* [Table_Size];
                for(int i = 0; i < Table_Size; i++) {
                    table[i] = NULL;
                }
            TS = Table_Size;
        }
        ~HashTable() {
            for(int i = 0; i < TS; i++) {
                if(table[i] != NULL) {
                    delete table[i];
                }
            }
            delete[] table;
        }
        //Initial hash method
        //Hash function is key mod table size.
        //Uses linear probing for collision avoidance.
        void hash_1(int key) {
            int hash_value = (key % TS);
            while(table[hash_value] != NULL && table[hash_value]->getKey() != key) {
                hash_value = (hash_value + 1) % TS;
            }
            if(table[hash_value] != NULL) {
                delete table[hash_value];
            }
            table[hash_value] = new Entry(key, hash_value);
        }
        //Print method
        void PrintTable() {
            cout << "Table Size: " << TS << endl;
            for(int i = 0; i < TS; i++) {
                if(table[i] != NULL) {
                    cout << "Key: " << table[i]->getKey() << "  Hash Value: " << table[i]->getHash() << endl;
                }
            }
        }
};
```

The 12-integer sequence was initialized inside the main function as an array. For part 1, the HashTable pointer HT1 was initialized, created, and printed.

Current Main Function and Output:



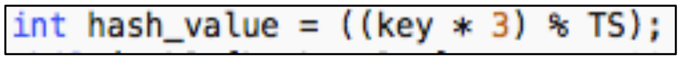
```
int main() {
    //12-integer sequence to be input to hash table.
    int numeros[] = {121, 81, 16, 100, 25, 0, 1, 9, 4, 36, 64, 49};
    HashTable *HT1 = new HashTable(17);
    for(int i = 0; i < 12; i++) {
        HT1->hash_1(numeros[i]);
    }
    HT1->PrintTable();
}
```



```
Ace:Unit 5 A$ g++ hash.cpp -o 5
Ace:Unit 5 A$ ./5
Table Size: 17
Key: 0  Hash Value: 0
Key: 1  Hash Value: 1
Key: 121  Hash Value: 2
Key: 36  Hash Value: 3
Key: 4  Hash Value: 4
Key: 49  Hash Value: 5
Key: 25  Hash Value: 8
Key: 9  Hash Value: 9
Key: 81  Hash Value: 13
Key: 64  Hash Value: 14
Key: 100  Hash Value: 15
Key: 16  Hash Value: 16
Ace:Unit 5 A$
```

2) For part 2, the goal is to implement rehashing for the same 12-integer sequence from part 1 starting with a reduced table size of 7. The first step was to edit the HashTable class as necessary to account for load factors and table resizing. The methods hash_2 and resize were added to the class. The hash_2 method calls the parameters of int keys[], which is the array of keys to be hashed and keys_count, which is the number of keys in the array. The method uses the same flow for entering entries to the hash table by calling the initial hash method, hash_1, to actually

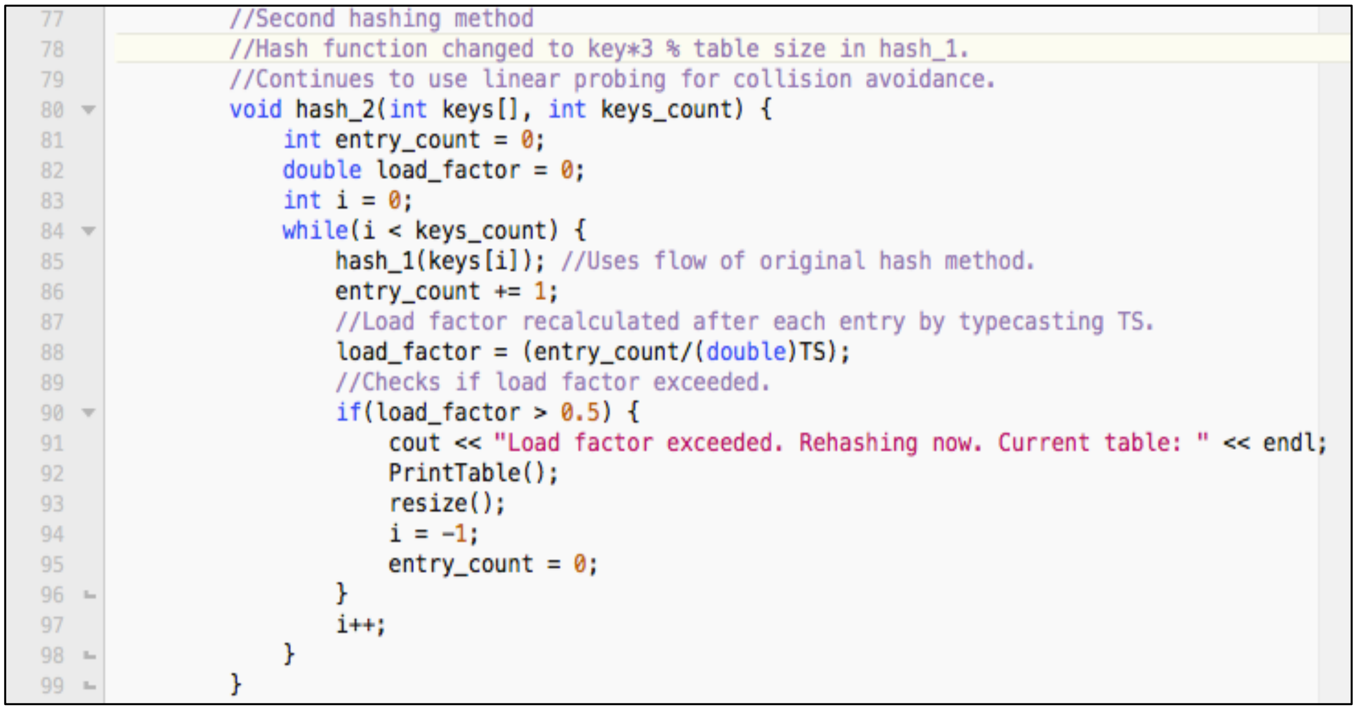
hash each key in the array. For part 2, the hash function was changed to ‘key * 3 mod table size’:



```
int hash_value = ((key * 3) % TS);
```

The hash_2 method also implements a load factor check as each entry is entered to the hash table and determines if rehashing is necessary. For this method, a commonly accepted load factor of 0.5 was used. If the load factor is exceeded, the method will print a notification and begin rehashing by calling the resize method. Once the resizing is complete, the entry count and iterator are reset, thus each key in the array of keys is rehashed from the beginning.

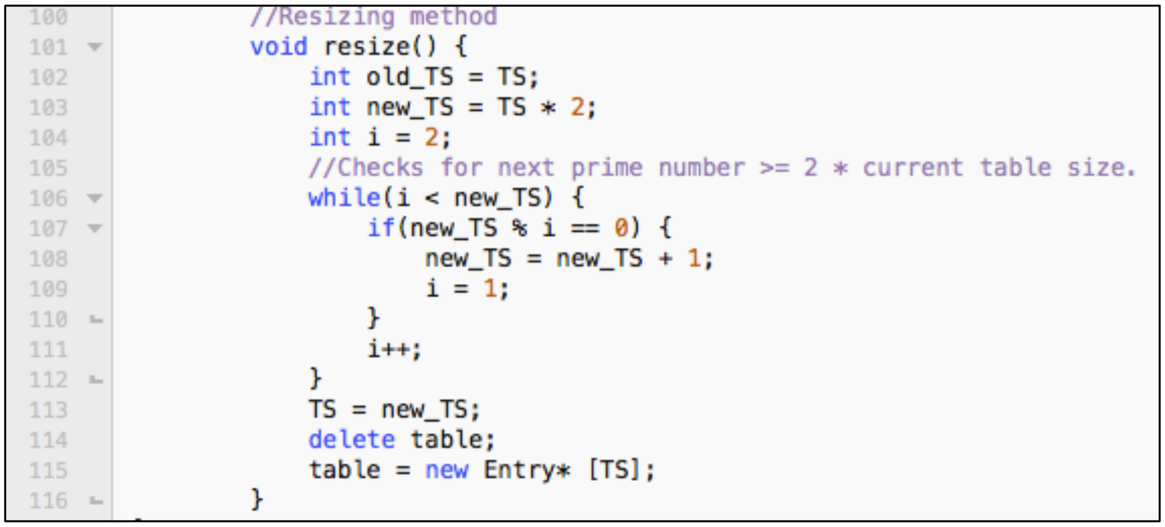
Hash_2 Method:



```
        //Second hashing method
        //Hash function changed to key*3 % table size in hash_1.
        //Continues to use linear probing for collision avoidance.
        void hash_2(int keys[], int keys_count) {
            int entry_count = 0;
            double load_factor = 0;
            int i = 0;
            while(i < keys_count) {
                hash_1(keys[i]); //Uses flow of original hash method.
                entry_count += 1;
                //Load factor recalculated after each entry by typecasting TS.
                load_factor = (entry_count/(double)TS);
                //Checks if load factor exceeded.
                if(load_factor > 0.5) {
                    cout << "Load factor exceeded. Rehashing now. Current table: " << endl;
                    PrintTable();
                    resize();
                    i = -1;
                    entry_count = 0;
                }
                i++;
            }
        }
```

The resizing method serves the purposes of determining the next appropriate table size and resizing the table attribute for the HashTable object. Doubling the current table size and then checking for the next prime number determines the next table size. In order to actually resize the table, the current table size is changed to the new table size. Then, the current table is deleted and a new table is generated using the updated table size.

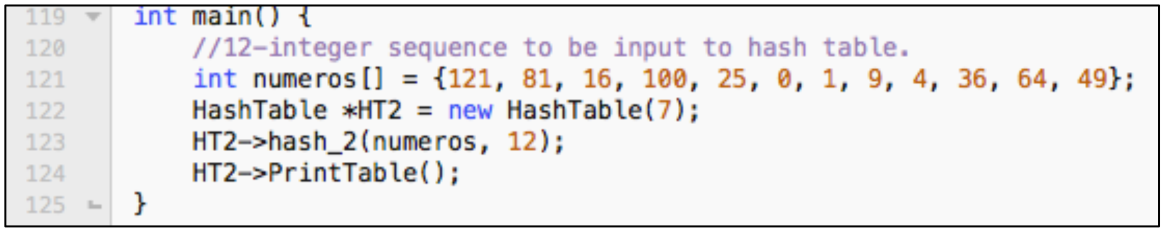
Resizing Method:



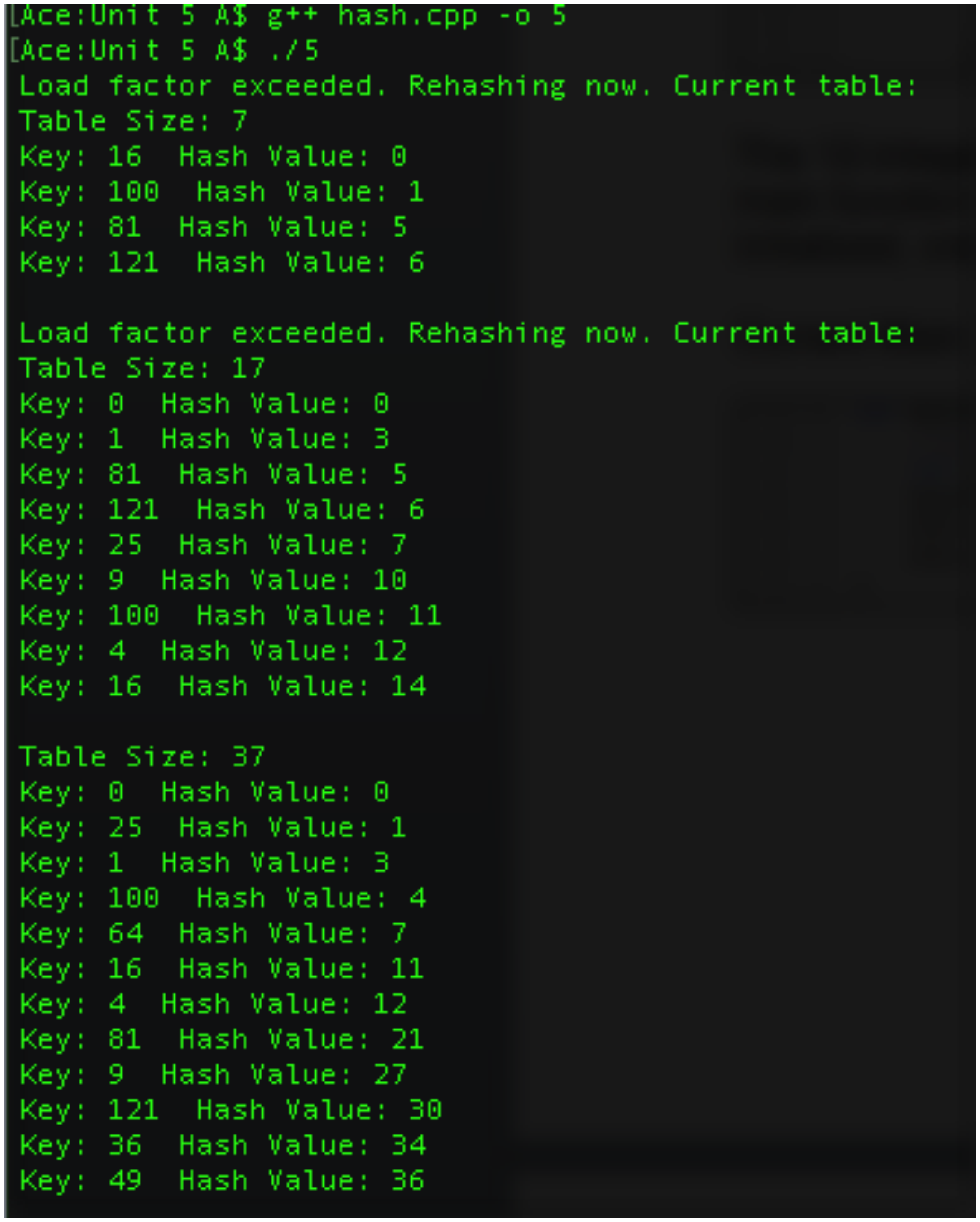
```
        //Resizing method
        void resize() {
            int old_TS = TS;
            int new_TS = TS * 2;
            int i = 2;
            //Checks for next prime number >= 2 * current table size.
            while(i < new_TS) {
                if(new_TS % i == 0) {
                    new_TS = new_TS + 1;
                    i = 1;
                }
                i++;
            }
            TS = new_TS;
            delete table;
            table = new Entry* [TS];
        }
```

The 12-integer sequence of keys was again initialized in the main function. For part 2, HashTable pointer HT2 was initialized, created, and printed. Based on the load factor used, the initial table was rehashed twice. The intermediate rehashed table used a table size of 17 and the final rehashed table used a table size of 37.

Current Main Function and Output:



```
int main() {
    //12-integer sequence to be input to hash table.
    int numeros[] = {121, 81, 16, 100, 25, 0, 1, 9, 4, 36, 64, 49};
    HashTable *HT2 = new HashTable(7);
    HT2->hash_2(numeros, 12);
    HT2->PrintTable();
}
```


```
Ace:Unit 5 A$ g++ hash.cpp -o 5
Ace:Unit 5 A$ ./5
Load factor exceeded. Rehashing now. Current table:
Table Size: 7
Key: 16  Hash Value: 0
Key: 100  Hash Value: 1
Key: 81  Hash Value: 5
Key: 121  Hash Value: 6

Load factor exceeded. Rehashing now. Current table:
Table Size: 17
Key: 0  Hash Value: 0
Key: 1  Hash Value: 3
Key: 81  Hash Value: 5
Key: 121  Hash Value: 6
Key: 25  Hash Value: 7
Key: 9  Hash Value: 10
Key: 100  Hash Value: 11
Key: 4  Hash Value: 12
Key: 16  Hash Value: 14

Table Size: 37
Key: 0  Hash Value: 0
Key: 25  Hash Value: 1
Key: 1  Hash Value: 3
Key: 100  Hash Value: 4
Key: 64  Hash Value: 7
Key: 16  Hash Value: 11
Key: 4  Hash Value: 12
Key: 81  Hash Value: 21
Key: 9  Hash Value: 27
Key: 121  Hash Value: 30
Key: 36  Hash Value: 34
Key: 49  Hash Value: 36
```