DNSRecord.cpp

```
#include <iostream>
#include "DNSRecord.h"

DNSRecord::DNSRecord(const std::string& ipAddress, int ttl) : ipAddress(ipAddress) {
    expirationTime = time(nullptr) + ttl;
}

bool DNSRecord::isExpired() {
    return time(nullptr) >= expirationTime;
}

std::string DNSRecord::getIPAddress() {
    return ipAddress;
}

std::ostream& operator<<(std::ostream& os, const DNSRecord& record) {
    os << "DNSRecord(ip_address='" << record.ipAddress << "', expiration_time=" << r
ecord.expirationTime << ")";
    return os;
}
```

DNSRecord.h

```
#ifndef DNSRECORD_H
#define DNSRECORD_H

#include <string>
#include <ctime>

class DNSRecord {
public:
    DNSRecord(const std::string& ipAddress, int ttl);

    bool isExpired();
    std::string getIPAddress();

    friend std::ostream& operator<<(std::ostream& os, const DNSRecord& record);

private:
    std::string ipAddress;
    time_t expirationTime;
};

#endif  // DNSRECORD_H
```

HashTable.cpp

```
#include <iostream>
#include "HashTable.h"
```

```
HashTable::HashTable(int size) : size(size), table(size, nullptr) {}

void HashTable::insert(const std::string& key, const DNSRecord& record) {

}

std::string HashTable::get(const std::string& key) {
    int hashValue = hash(key) % size;
    Node* node = table[hashValue];

    while (node != nullptr) {
        if (node->key == key) {
            if (node->record.isExpired()) {
                remove(key);
                return "None";
            }
            return node->record.getIPAddress();
        }
        node = node->next;
    }

    return "None";
}

void HashTable::remove(const std::string& key) {

}

void HashTable::display() {

}

int HashTable::hash(const std::string& key) {

}
```

HashTable.h

```
#ifndef HASHTABLE_H
#define HASHTABLE_H

#include <vector>
#include "Node.h"

class HashTable {
public:
    HashTable(int size);

    void insert(const std::string& key, const DNSRecord& record);
```

```
    std::string get(const std::string& key);
    void remove(const std::string& key);
    void display();

private:
    int size;
    std::vector<Node*> table;
    std::hash<std::string> hashFunction;

    int hash(const std::string& key);
};

#endif  // HASHTABLE_H
```

Makefile

```
CC = g++
CFLAGS = -std=c++11

all: main
	@echo "Build complete."

main: main.o HashTable.o DNSRecord.o Node.o
	$(CC) $(CFLAGS) main.o HashTable.o DNSRecord.o Node.o -o main

main.o: main.cpp
	$(CC) $(CFLAGS) -c main.cpp

HashTable.o: HashTable.cpp HashTable.h Node.h DNSRecord.h
	$(CC) $(CFLAGS) -c HashTable.cpp

DNSRecord.o: DNSRecord.cpp DNSRecord.h
	$(CC) $(CFLAGS) -c DNSRecord.cpp

Node.o: Node.cpp Node.h DNSRecord.h
	$(CC) $(CFLAGS) -c Node.cpp

clean:
	rm -f *.o main
```

Node.cpp

```
#include "Node.h"

Node::Node(const std::string& key, const DNSRecord& record) : key(key), record(record),
next(nullptr) {}
```

Node.h

```
#ifndef NODE_H
#define NODE_H

#include <string>
#include "DNSRecord.h"

class Node {
public:
    Node(const std::string& key, const DNSRecord& record);

    std::string key;
    DNSRecord record;
    Node* next;
};

#endif  // NODE_H
```

Main.cpp

```
#include <iostream>
#include <unistd.h>
#include "HashTable.h"
#include "DNSRecord.h"

int main() {
    HashTable hashTable(10);

    hashTable.insert("www.example.com", DNSRecord("192.168.1.100", 5));  // Record expires in
60 seconds

    std::cout << hashTable.get("www.example.com") << std::endl;  // Output: 192.168.1.100

    sleep(6);  // Sleep for more than the record's TTL to simulate expiration

    std::cout << hashTable.get("www.example.com") << std::endl;  // Output: None

    hashTable.display();

    return 0;
}
```

Rubric.txt

Hash Table Implementation: (25 points):

Each bucket in the hash table should be implemented as a linked list to handle collisions. (10):
Use a custom hash function to calculate the hash value for domain name keys. (15):

Functionality (35):

Include functionalities to insert domain name and IP address pairs into the DNS cache. (10):
Each entry should include an expiration time (TTL): received from the DNS server. (10):
Implement a lookup function to retrieve IP addresses based on domain names. (5):
If a requested domain name is found and has not expired, return the corresponding IP address. (5):
Provide a display function to show the contents of the hash table, including the linked list within each bucket. (5):

Code Quality (10 points):

Memory management, no memory leaks (10 points)::
Code organization and readability, proper naming conventions, comments, and indentation (5 points)::

Test Cases (20 points):

Test coverage -- comprehensive set of test cases to cover different scenarios and functionalities (10 points)::
Test accuracy -- test cases correctly evaluate the expected behavior of the system (10 points)::

Readme / Documentation (10 points):

Readme is updated under the picture to explain challeneges in the project, how to run the code, and a description of the workflow (15 points)::

Total Possible Points: 100
Student Actual Points:

README.md

Project Description: DNS Cache using Hash Table

This project implements a DNS (Domain Name System) cache using a hash table data structure in C++. The DNS cache is a crucial component of networking systems as it stores recently resolved domain names and their corresponding IP addresses, enabling faster future lookups.

The project consists of the following components:

Hash Table: The hash table is the core data structure used to store the DNS cache. It is implemented as an array of linked lists. Each bucket in the hash table represents a linked list of key-value pairs, where the key is the domain name, and the value is the associated IP address and expiration time.

DNSRecord: The DNSRecord class represents a record in the DNS cache. It contains the IP address and expiration time of the record. The expiration time is set based on the TTL (Time-to-Live) value received from the DNS server. The DNSRecord class also provides a method to check if a record has expired.

Node: The Node class represents a node in the linked list within each bucket of the hash table. Each node holds a key-value pair, where the key is the domain name, and the value is a DNSRecord object.

Hash Function: A hash function is used to map the domain name keys to an index in the hash table. In this project, a custom hash function based on the std::hash is employed.

The project allows the following functionalities:

Insertion: The project enables the insertion of domain name and IP address pairs into the DNS cache. Each entry includes an expiration time (TTL) received from the DNS server.

Lookup: The project supports looking up IP addresses based on the domain name. If a requested domain name is found in the DNS cache and has not expired, the corresponding IP address is returned. Otherwise, if the entry has expired or is not present, "None" is returned.

Expiration: The project automatically removes expired entries from the DNS cache when they are looked up or accessed. This ensures that only valid and up-to-date entries are stored in the cache.

Display: The project provides a display function to show the contents of the hash table, including the linked list within each bucket. This facilitates understanding and monitoring the DNS cache.

The project is organized into multiple files, including main.cpp, HashTable.h, HashTable.cpp, DNSRecord.h, DNSRecord.cpp, Node.h, and Node.cpp. The Makefile is also provided to compile the project.

By implementing this DNS cache using a hash table, the project demonstrates an efficient approach to store and retrieve frequently accessed domain names and their corresponding IP addresses, reducing the need for frequent DNS lookups.