CA 3 PROGRAMMING PARADIGM

Group Members:

Sr. No.	PRN	Name of Student	Mail id
1	22070122047	OMKAR DAREKAR	omkar.darekar.btech2022@sitpune.edu.in
2	22070122035	ARYAN BACHUTE	aryan.bachute.btech2022@sitpune.edu.in
3	22070122081	HRIDAY THAKER	hriday.thaker.btech2022@sitpune.edu.in

PROBLEM STATEMENT

Modern computer systems use caches to accelerate access to frequently used data. Cache replacement policies determine which data is retained in the cache when new data needs to be loaded, and the choice of policy can significantly impact system performance. This project aims to develop a cache replacement policy simulator in C++ to understand and compare the behavior of two popular cache replacement policies: LRU (Least Recently Used) and LFU (Least Frequently Used), while incorporating the Principle of Locality.

EXPLANATION

Code Structure:

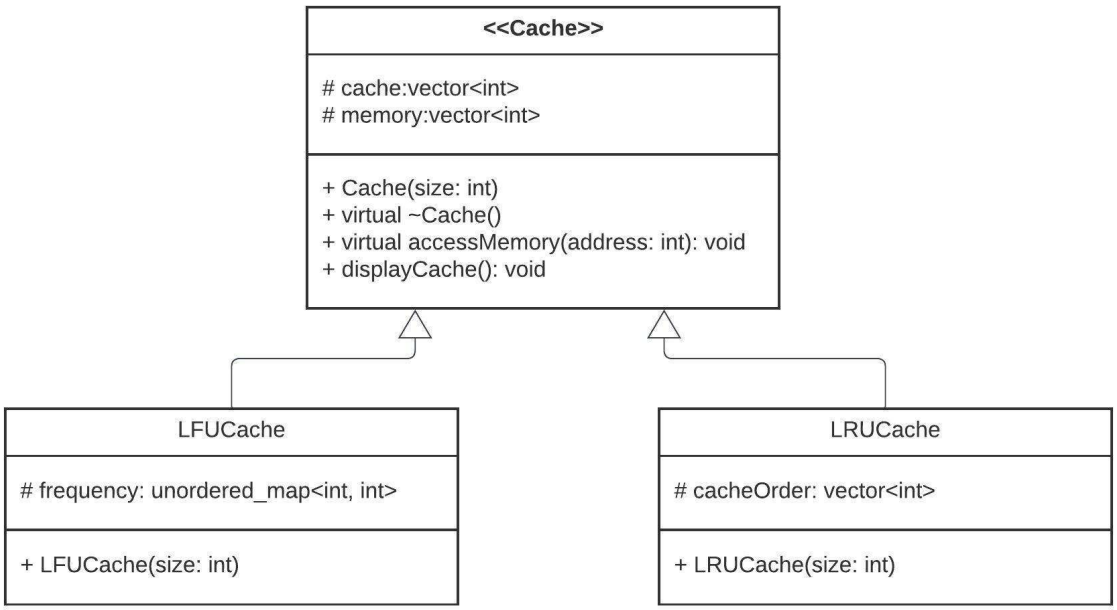
- Base Class (Cache): Serves as the foundation for cache management and provides shared functionality for the cache system.
- LRUCache Class (Inherits from Cache): Demonstrates inheritance, extending the base class to implement the LRU (Least Recently Used) replacement policy. This class showcases the concept of inheritance, where it inherits the attributes and behaviors of the base class, Cache.
- LFUCache Class (Inherits from Cache): Similarly, this class illustrates inheritance as it extends the Cache class to implement the LFU (Least Frequently Used) replacement policy. In both derived classes, inheritance allows reusing common cache-related attributes and methods from the base class.
- Main Function demonstrating polymorphism and inheritance: In the main function, polymorphism and inheritance work together seamlessly. The program interacts with objects of the base class Cache and its derived classes (LRUCache and LFUCache). This dynamic behavior allows switching between different cache replacement policies (LRU and LFU) through the base class pointer (cachePolicy). The program showcases how inheritance enables the derived classes to inherit common attributes and methods from the base class. Through polymorphism, the program invokes the appropriate overridden accessMemory function, specific to the selected cache policy, at runtime. This design ensures flexibility and modularity in cache management, facilitating easy extension to other cache policies.

Cache Policies:

- LRU: Removes the least recently accessed memory address when the cache is full.
- LFU: Evicts the memory address with the lowest access frequency when the cache is full.

Principle of Locality:
The code implements the Principle of Locality, which enhances cache performance by storing nearby memory addresses in the cache, minimizing cache misses.

CLASS DIAGRAM:

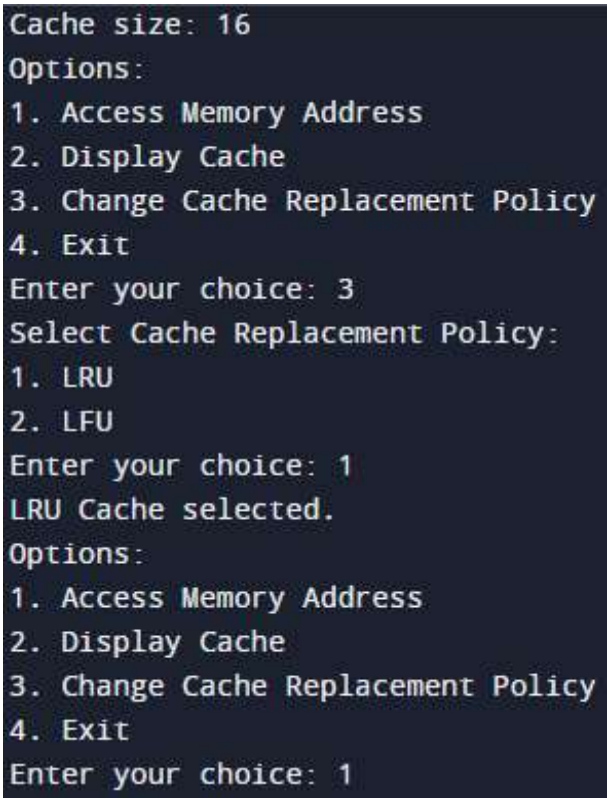


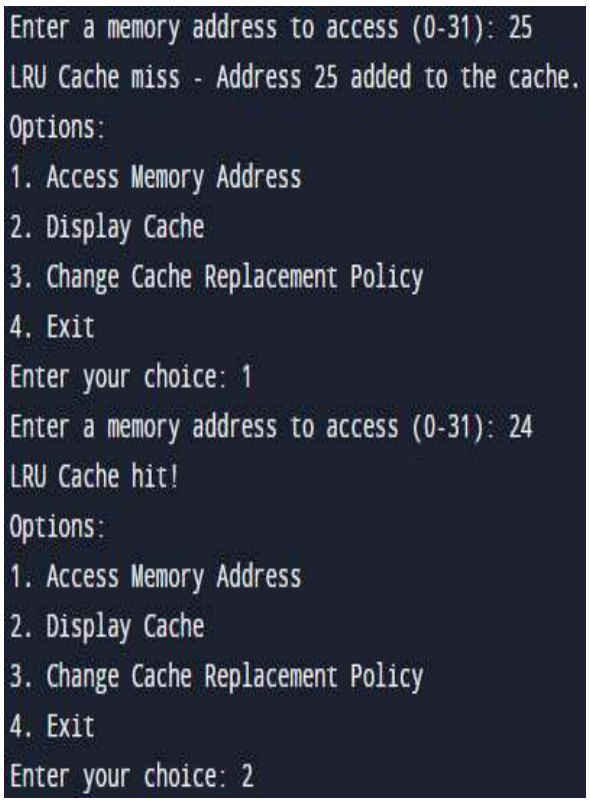
CODE SNIPPETS:

```
void accessMemory(int address) override//overrides base class
{
    auto it = find(cache.begin(), cache.end(), address);//search address in cache
    if (it != cache.end())//if addres found then it will not reach cache end
    {
        cout << "LRU Cache hit!" << endl;
        cacheOrder.erase(remove(cacheOrder.begin(), cacheOrder.end(), address), cacheOrder.end
            ());
        cacheOrder.push_back(address);//keeping the order of cache updated
    }
```

```
void accessMemory(int address) override
{
    auto it = find(cache.begin(), cache.end(), address);
    if (it != cache.end())
    {
        cout << "LFU Cache hit!" << endl;//address found in the cache
        frequency[address]++;
    }
    else
```

INPUT/OUTPUT





GITHUB REPOSITORY LINK:

https://github.com/thakerhriday/Programming_Paradigms