

**California State University, Fresno**  
**Lyles College of Engineering**  
**Electrical and Computer Engineering Department**

**PROJECT 2 REPORT**

**Assignment:**            **Project 2**

**Experiment Title:**    **Simulating Cache Replacement Schemes using C++**

**Course Title:**        **ECE 144 (Embedded Operating Systems)**

**Instructor:**           **Dr. Nan Wang**

**Prepared by:** Aryan Singh

Student ID # 300404053

**Date Submitted:** 11/13/2023

**INSTRUCTOR SECTION**

**Comments:** \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Final Grade:** \_\_\_\_\_

# Table of Contents

<b>1. Objective .....</b>	<b>2</b>
<b>2. Hardware Requirements .....</b>	<b>2</b>
<b>3. Software Requirements .....</b>	<b>2</b>
<b>4. Introduction .....</b>	<b>2</b>
<b>5. Problem Statement .....</b>	<b>3</b>
<b>6. Implementation .....</b>	<b>4</b>
<b>7. Result and Comparison.....</b>	<b>7</b>
<b>8. Conclusion .....</b>	<b>13</b>
<b>9. References .....</b>	<b>14</b>
<b>10. Appendix .....</b>	<b>14</b>

## 1. Objective

The objective of this assignment is to gain understanding of the page replacement algorithms that are used by the Operating System when a page fault occurs. When a page fault occurs, the operating system has to choose a page from the virtual memory in cache, to remove from the memory to make room for the incoming new page. There are several page replacement schemes that are used to perform the aforementioned, however, through this project we will be realizing how to implement the FIFO (First in First out), LRU (Least Recently Used) and Radom replacement schemes through the use of C++ coding language.

## 2. Hardware Requirements

- Personal Computer that supports C++ software (Visual Studio)

## 3. Software Requirements

- Visual Studio C++ programming tool
- Snipping Tool
- Microsoft Word or comparable Word Processor

## 4. Introduction

When a Page Fault occurs, the Operating System uses the Page Replacement Technique to manage memory in a computer system. Page replacement is done in order to overcome the

shortcomings of physical memory. The physical memory or RAM is limited in size and cannot comprehend all the processes and data running on an Operating System. So, virtual memory is used which acts as an extension of the physical memory. Pages, which are basically blocks of virtual memory are used for more efficient memory utilization. Also, since the memory fetching from the physical memory is a very slow process, so caches are used which are a smaller and faster storage location that temporarily stores copies of recently used or most used data.

Therefore, in the case of page faults from the cache, the page replacement occurs in the cache according to the three page replacement techniques. The FIFO (First In First Out) page algorithm, the LRU (Least Recently Used) algorithm and the Random page replacement algorithm are the three page replacement techniques.

In FIFO algorithm, the Operating system maintains a list of all pages currently in memory, with the page at the head of the list the oldest one and the page at the tail the most recent arrival. When a page fault occurs, then the page at the head is removed and the new page is added at the end of the list.

In LRU algorithm, the pages are set the same way as they were for FIFO, but in this case the page that has not been accessed for the longest time will be removed and the new incoming page will be placed in the place of the least recently used page.

For the case of Random page replacement algorithm, when a page fault occurs then a random page from the cache is replaced with the new page.

## **5. Problem Statement**

In a virtual memory system, when a page fault occurs, it decreases the efficiency of the operating system and increase the latency. Page faults occur when a program tries to access a page that is not present in cache memory, and since the page is not present there then the program looks into the main memory which is a very slow process. So, the problem occurs when this kind of situation arises. In order to keep the efficiency of the program high and the latency low, the page replacement techniques are used.

## 6. Implementation

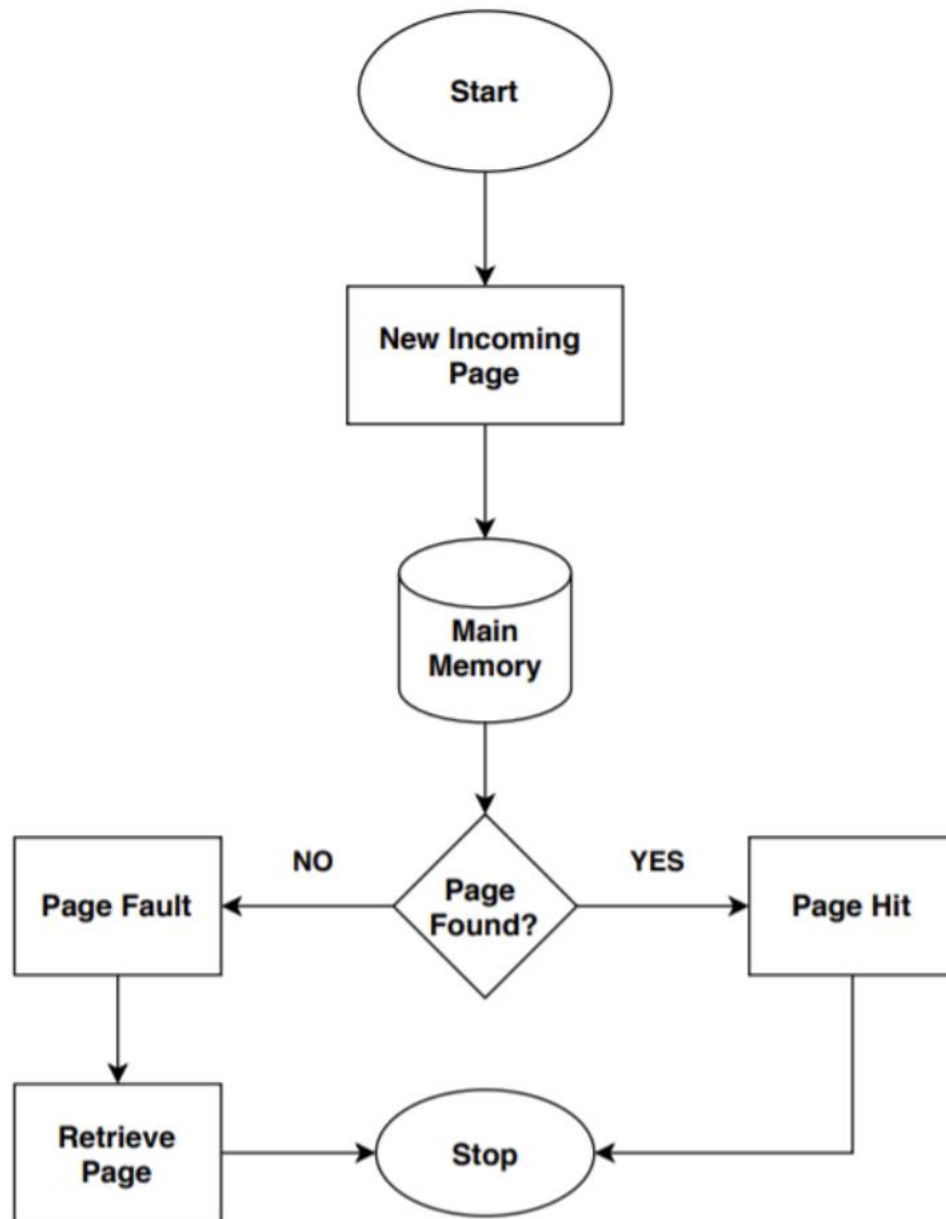


Figure 1: Flow Diagram for the logic behind page replacement algorithms

To implement the page replacement techniques the process flow layout is similar to Figure 1. For the implementation of the FIFO (First in First out) page replacement algorithm, the C++ code is present in Figure 2. The implementation of the LRU (Least Recently Used) replacement algorithm can be seen in Figure 3 and the implementation of the Randomly generated page replacement was shown in Figure 4.

```

9 int simulateFIFO(const int* pageReferences, int pageReferencesSize, int cacheSize) {
10     std::deque<int> cache;
11     int cacheHits = 0;
12
13     for (int i = 0; i < pageReferencesSize; i++) {
14         int page = pageReferences[i];
15         bool found = false;
16
17         // Search for 'page' in the cache
18         for (int j = 0; j < cache.size(); j++) {
19             if (cache[j] == page) {
20                 found = true;
21                 break;
22             }
23         }
24
25         if (found) {
26             cacheHits++;
27         } else {
28             if (cache.size() == cacheSize) {
29                 cache.pop_front();
30             }
31             cache.push_back(page);
32         }
33     }
34
35     return cacheHits;
36 }
37

```

Figure 2: FIFO implementation using C++ Code

```

63 int simulateLRU(const int* pageReferences, int pageReferencesSize, int cacheSize)
64 {   std::deque<int> cache;
65     int cacheHits = 0;
66     for (int i = 0; i < pageReferencesSize; i++) {
67         int page = pageReferences[i];
68         bool found = false;
69
70         // Search for 'page' in the cache
71         for (int j = 0; j < cache.size(); j++) {
72             if (cache[j] == page) {
73                 found = true;
74                 // Move the found page to the back to represent it as the most recently used
75                 int temp = cache[j];
76                 cache.erase(cache.begin() + j);
77                 cache.push_back(temp);
78                 break;
79             }
80         }
81
82         if (found) {
83             cacheHits++;
84         } else {
85             if (cache.size() == cacheSize) {
86                 cache.pop_front();
87             }
88             cache.push_back(page);
89         }
90     }
91
92     return cacheHits;
93 }

```

Figure 3: Implementing LRU (Least recently used) algorithm using C++ code

```

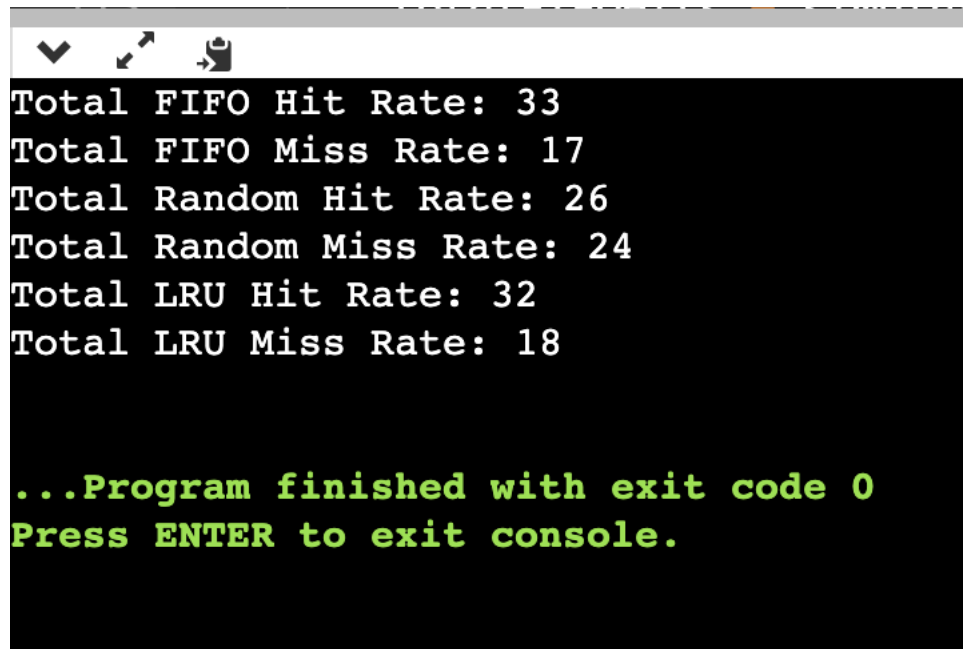
41 int simulateRandom(const int* pageReferences, int pageReferencesSize, int cacheSize) {
42     std::unordered_set<int> cache;
43     int cacheHits = 0;
44
45     for (int i = 0; i < pageReferencesSize; i++) {
46         int page = pageReferences[i];
47         if (cache.find(page) != cache.end()) {
48             cacheHits++;
49         } else {
50             if (cache.size() == cacheSize) {
51                 int randomIndex = rand() % cacheSize;
52                 auto it = cache.begin();
53                 std::advance(it, randomIndex);
54                 cache.erase(it);
55             }
56             cache.insert(page);
57         }
58     }
59
60     return cacheHits;
61 }
62

```

Figure 4: Random page replacement algorithm using C++ Code

## 7. Result and Comparison

The hit and miss rates for the 3 page replacement techniques can be seen in Figure 5. From the results it can be seen that the FIFO replacement technique had the most hit rates so using the FIFO technique proved to be the most efficient on the given set of array of data. The LRU replacement technique was the second best with the random replacement technique showing the worst results in all cases. The working code for the whole C++ program can be seen in Figures 6 through 10.

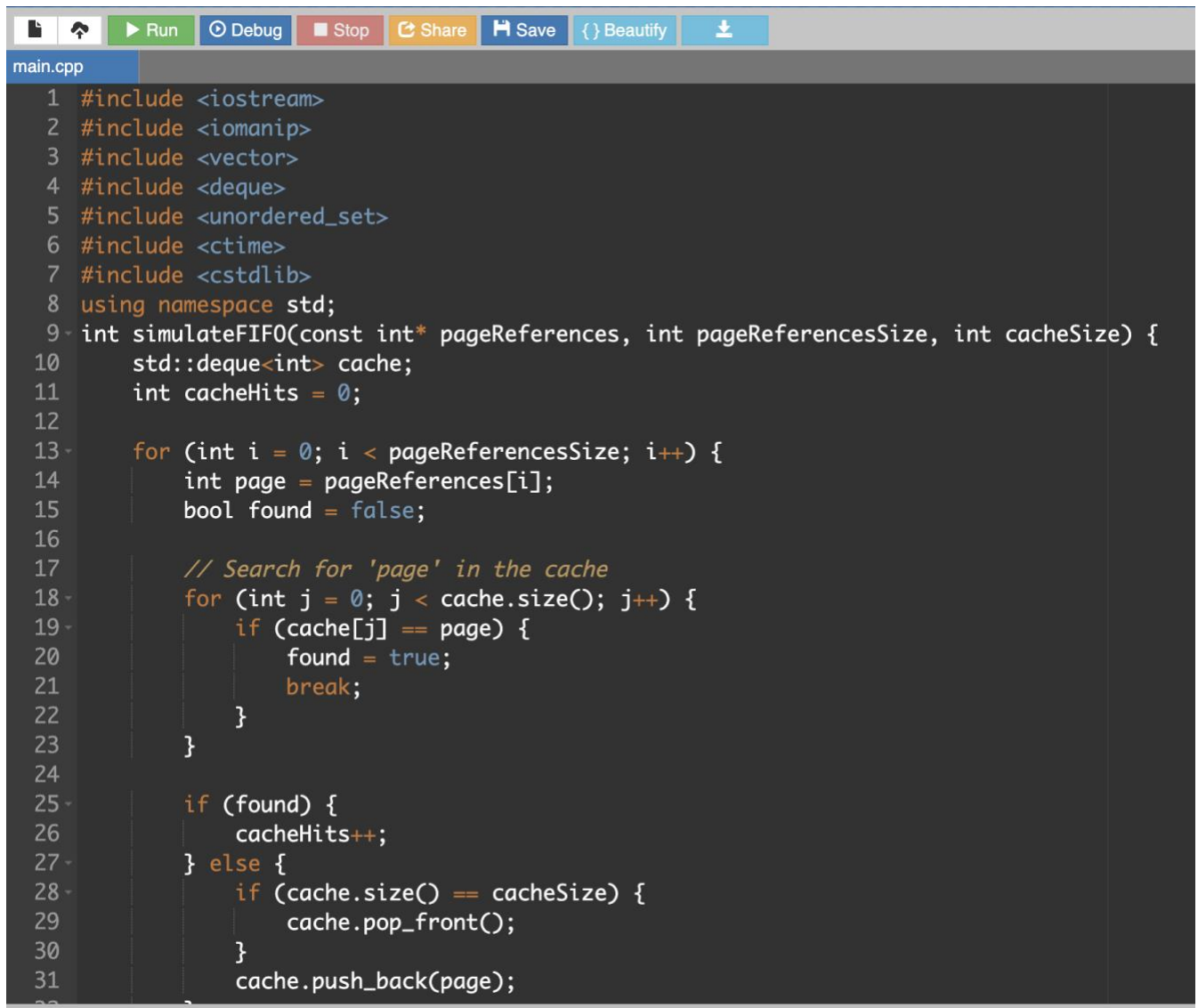


```
✓ ↗ 📄
Total FIFO Hit Rate: 33
Total FIFO Miss Rate: 17
Total Random Hit Rate: 26
Total Random Miss Rate: 24
Total LRU Hit Rate: 32
Total LRU Miss Rate: 18

...Program finished with exit code 0
Press ENTER to exit console.
```

Figure 5: Output for the hit and miss rates for all 3 replacement algorithms





```
1 #include <iostream>
2 #include <iomanip>
3 #include <vector>
4 #include <deque>
5 #include <unordered_set>
6 #include <ctime>
7 #include <cstdlib>
8 using namespace std;
9 int simulateFIFO(const int* pageReferences, int pageReferencesSize, int cacheSize) {
10     std::deque<int> cache;
11     int cacheHits = 0;
12
13     for (int i = 0; i < pageReferencesSize; i++) {
14         int page = pageReferences[i];
15         bool found = false;
16
17         // Search for 'page' in the cache
18         for (int j = 0; j < cache.size(); j++) {
19             if (cache[j] == page) {
20                 found = true;
21                 break;
22             }
23         }
24
25         if (found) {
26             cacheHits++;
27         } else {
28             if (cache.size() == cacheSize) {
29                 cache.pop_front();
30             }
31             cache.push_back(page);
32         }
33     }
```

Figure 6: Entire C++ Code for page replacement implementation (1)

```

main.cpp
29         cache.pop_front();
30     }
31     cache.push_back(page);
32 }
33 }
34
35     return cacheHits;
36 }
37
38
39
40
41 int simulateRandom(const int* pageReferences, int pageReferencesSize, int cacheSize) {
42     std::unordered_set<int> cache;
43     int cacheHits = 0;
44
45     for (int i = 0; i < pageReferencesSize; i++) {
46         int page = pageReferences[i];
47         if (cache.find(page) != cache.end()) {
48             cacheHits++;
49         } else {
50             if (cache.size() == cacheSize) {
51                 int randomIndex = rand() % cacheSize;
52                 auto it = cache.begin();
53                 std::advance(it, randomIndex);
54                 cache.erase(it);
55             }
56             cache.insert(page);
57         }
58     }
59
60     return cacheHits;

```

Figure 7: Entire C++ Code for page replacement implementation (2)

```

main.cpp
56     cache.insert(page);
57 }
58 }
59
60 return cacheHits;
61 }
62
63 int simulateLRU(const int* pageReferences, int pageReferencesSize, int cacheSize)
64 {
65     std::deque<int> cache;
66     int cacheHits = 0;
67     for (int i = 0; i < pageReferencesSize; i++) {
68         int page = pageReferences[i];
69         bool found = false;
70
71         // Search for 'page' in the cache
72         for (int j = 0; j < cache.size(); j++) {
73             if (cache[j] == page) {
74                 found = true;
75                 // Move the found page to the back to represent it as the most recently used
76                 int temp = cache[j];
77                 cache.erase(cache.begin() + j);
78                 cache.push_back(temp);
79                 break;
80             }
81         }
82
83         if (found) {
84             cacheHits++;
85         } else {
86             if (cache.size() == cacheSize) {
87                 cache.pop_front();
88             }
89             cache.push_back(page);
90         }
91     }
92 }

```

Figure 8: Entire C++ Code for page replacement implementation (3)

```

main.cpp
85         if (cache.size() == cacheSize) {
86             cache.pop_front();
87         }
88         cache.push_back(page);
89     }
90 }
91
92 return cacheHits;
93 }
94
95
96 int main() {
97     int array_Pages[] = {49, 50, 50, 52, 30, 49, 43, 49, 20, 31, 26, 37,
98                          43, 43, 43, 43, 43, 20, 20, 30, 21, 25, 25, 25,
99                          25, 25, 25, 31, 20, 20, 11, 11, 11, 12, 12, 49,
100                         43, 50, 50, 20, 11, 43, 50, 12, 12, 49, 43, 50, 50, 50};
101
102
103     int pageReferencesSize = sizeof(array_Pages) / sizeof(array_Pages[0]);
104     int cacheSize = 6;
105
106
107     int totalFIFOHits = 0;
108     int totalRandomHits = 0;
109     int totalLRUHits = 0;
110
111     srand(static_cast<unsigned int>(time(NULL)));
112
113     totalFIFOHits = simulateFIFO(array_Pages, pageReferencesSize, cacheSize);
114     totalRandomHits = simulateRandom(array_Pages, pageReferencesSize, cacheSize);
115     totalLRUHits = simulateLRU(array_Pages, pageReferencesSize, cacheSize);

```

Figure 9: Entire C++ Code for page replacement implementation (4)

```

main.cpp
100         43, 50, 50, 20, 11, 43, 50, 12, 12, 49, 43, 50, 50, 50};
101
102
103     int pageReferencesSize = sizeof(array_Pages) / sizeof(array_Pages[0]);
104     int cacheSize = 6;
105
106
107     int totalFIFOHits = 0;
108     int totalRandomHits = 0;
109     int totalLRUHits = 0;
110
111     srand(static_cast<unsigned int>(time(NULL)));
112
113     totalFIFOHits = simulateFIFO(array_Pages, pageReferencesSize, cacheSize);
114     totalRandomHits = simulateRandom(array_Pages, pageReferencesSize, cacheSize);
115     totalLRUHits = simulateLRU(array_Pages, pageReferencesSize, cacheSize);
116
117     int FIFOMiss = 50 - totalFIFOHits;
118     int LRUMiss = 50 - totalLRUHits;
119     int RandomMiss = 50 - totalRandomHits;
120
121
122     std::cout << "Total FIFO Hit Rate: " << totalFIFOHits << std::endl;
123     std::cout << "Total FIFO Miss Rate: " << FIFOMiss << std::endl;
124
125     std::cout << "Total Random Hit Rate: " << totalRandomHits << std::endl;
126     std::cout << "Total Random Miss Rate: " << RandomMiss << std::endl;
127
128     std::cout << "Total LRU Hit Rate: " << totalLRUHits << std::endl;
129     std::cout << "Total LRU Miss Rate: " << LRUMiss << std::endl;
130
131     return 0;
132 }
133

```

Figure 10: Entire C++ Code for page replacement implementation (5)

## 8. Conclusion

Through this project we were able to understand the solution for the page fault problem in the Operating system and implement page replacement strategies in order to improve the efficiency and reduce latency in Operating System processes. We learned how to perform the 3 page replacement strategies using C++ and understood which one of them had the most efficiency.

## 9. References

Archiveddocs. (n.d.). *How to: Add items to the Cache*. Microsoft Learn.

[https://learn.microsoft.com/en-us/previous-versions/aspnet/18c1wd61\(v=vs.100\)](https://learn.microsoft.com/en-us/previous-versions/aspnet/18c1wd61(v=vs.100))

*Std::deque*. cppreference.com. (n.d.). <https://en.cppreference.com/w/cpp/container/deque>

W11. 18. Ch.4 Memory -3.pptx

[https://fresnostate.instructure.com/courses/74740/files/12651000?module\\_item\\_id=3595020](https://fresnostate.instructure.com/courses/74740/files/12651000?module_item_id=3595020)

## 10. Appendix

A. Code for C++ Program

```
#include <iostream>
```

```
#include <iomanip>
```

```
#include <vector>
```

```
#include <deque>
```

```
#include <unordered_set>
```

```
#include <ctime>
```

```
#include <cstdlib>
```

```
using namespace std;
```

```
int simulateFIFO(const int* pageReferences, int pageReferencesSize, int cacheSize) {
```

```
    std::deque<int> cache;
```

```
    int cacheHits = 0;
```

```
    for (int i = 0; i < pageReferencesSize; i++) {
```

```
        int page = pageReferences[i];
```

```

    bool found = false;

    // Search for 'page' in the cache
    for (int j = 0; j < cache.size(); j++) {
        if (cache[j] == page) {
            found = true;
            break;
        }
    }

    if (found) {
        cacheHits++;
    } else {
        if (cache.size() == cacheSize) {
            cache.pop_front();
        }
        cache.push_back(page);
    }
}

return cacheHits;
}

```

```

int simulateRandom(const int* pageReferences, int pageReferencesSize, int cacheSize) {

    std::unordered_set<int> cache;

    int cacheHits = 0;

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

        int page = pageReferences[i];

        if (cache.find(page) != cache.end()) {

            cacheHits++;

        } else {

            if (cache.size() == cacheSize) {

                int randomIndex = rand() % cacheSize;

                auto it = cache.begin();

                std::advance(it, randomIndex);

                cache.erase(it);

            }

            cache.insert(page);

        }

    }

    return cacheHits;
}

```



```
}
```

```
int simulateLRU(const int* pageReferences, int pageReferencesSize, int cacheSize)
```

```
{  std::deque<int> cache;
```

```
    int cacheHits = 0;
```

```
    for (int i = 0; i < pageReferencesSize; i++) {
```

```
        int page = pageReferences[i];
```

```
        bool found = false;
```

```
        // Search for 'page' in the cache
```

```
        for (int j = 0; j < cache.size(); j++) {
```

```
            if (cache[j] == page) {
```

```
                found = true;
```

```
                // Move the found page to the back to represent it as the most recently used
```

```
                int temp = cache[j];
```

```
                cache.erase(cache.begin() + j);
```

```
                cache.push_back(temp);
```

```
                break;
```

```
            }
```

```
        }
```

```
    if (found) {
```

```
        cacheHits++;
```

```

    } else {

        if (cache.size() == cacheSize) {

            cache.pop_front();

        }

        cache.push_back(page);

    }

}

return cacheHits;

}

int main() {

    int array_Pages[] = {49, 50, 50, 52, 30, 49, 43, 49, 20, 31, 26, 37,

        43, 43, 43, 43, 43, 20, 20, 30, 21, 25, 25, 25,

        25, 25, 25, 31, 20, 20, 11, 11, 11, 12, 12, 49,

        43, 50, 50, 20, 11, 43, 50, 12, 12, 49, 43, 50, 50, 50};

    int pageReferencesSize = sizeof(array_Pages) / sizeof(array_Pages[0]);

    int cacheSize = 6;

    int totalFIFOHits = 0;

    int totalRandomHits = 0;

    int totalLRUHits = 0;

```

```

srand(static_cast<unsigned int>(time(NULL)));

totalFIFOHits = simulateFIFO(array_Pages, pageReferencesSize, cacheSize);
totalRandomHits = simulateRandom(array_Pages, pageReferencesSize, cacheSize);
totalLRUHits = simulateLRU(array_Pages, pageReferencesSize, cacheSize);

int FIFOMiss = 50 - totalFIFOHits;
int LRUmiss = 50 - totalLRUHits;
int Randommiss = 50 - totalRandomHits;

std::cout << "Total FIFO Hit Rate: " << totalFIFOHits << std::endl;
std::cout << "Total FIFO Miss Rate: " << FIFOMiss << std::endl;

std::cout << "Total Random Hit Rate: " << totalRandomHits << std::endl;
std::cout << "Total Random Miss Rate: " << Randommiss << std::endl;

std::cout << "Total LRU Hit Rate: " << totalLRUHits << std::endl;
std::cout << "Total LRU Miss Rate: " << LRUmiss << endl;

return 0;
}

```