VMEMMAN Source Code

main.cpp:

```
#include <iostream>
#include <fstream>
#include<bits/stdc++.h>
#include "fifo.cpp"
#include "lru.cpp"
#include "mru.cpp"
#include "optimal.cpp"

using namespace std;

//Functions Prototype
int fifo(int pages[], float n, int frames); //fifo
int leastRecentlyUsed(int pages[], float n, int frames); //lru
int mostRecentlyUsed(int pages[], float n, int frames); //mru

//optimal functions
int predict(int pages[], vector<int>& fr, int n, int index);
bool search(int key, vector<int>& fr);
void opr(int pages[], float n, int frames);

//main function to run code
int main()
{
    //Declare and Initialize Variables
    ifstream data;
    data.open("data.txt");
    int arraySize;
    int pages[arraySize];   // Array of n elements
    int n = sizeof(pages)/sizeof(pages[0]);
        int frames;


    int i = 0;             // Loop counter variable
    ifstream inputFile;        // Input file stream object

    while (i < arraySize && data >> pages[i]){
        i++;
    }

    cout<<"Enter Page Size: "<<endl;
    cin>>n;
    cout<<"Enter # of Frames: "<<endl;
    cin>>frames;
    cout<<endl;
    cout<<"Page Size"<<"   "<<"# of Frames"<<"   "<<"Algorithm"<<"   "<<"
Page Fault %"<<endl;
    cout<<"************************************************"<<endl;
        fifo(pages, n, frames); ///call and output fifo
        leastRecentlyUsed(pages, n, frames);///call and output lru
        mostRecentlyUsed(pages, n, frames); ///call and output mru
    optimal(pages, n, frames);///call optimal function
```

```
    /// Outfile data
    ofstream outfile;
    outfile.open("outFile.txt", std::ios_base::app); // append instead of
overwrite
    outfile<<"Page Size"<<"   "<<"# of Frames"<<"   "<<"Algorithm"<<"   "<<"
Page Fault %"<<endl;
    outfile<<"*************************************************"<<endl;

    outfile.close();

        cout<<endl;

        //return 0;
}
```

fifo.cpp:

```
#include<bits/stdc++.h>
#include <iostream>
#include <fstream>

using namespace std;
/// FIFO function to find page faults
int fifo(int pages[], float n, int frames)
{
        /// Set of current pages to check if pages are present or not
        unordered_set<int> s;

        /// Store the pages using FIFO
        queue<int> indexes;

        /// Initialize and loop to iterate through indicated frame size, n.
        float page_faults = 0;
        for (int i=0; i<n; i++)
        {
                /// Check if the set can store more pages
                if (s.size() < frames)
                {
                        /// Insert page into set if not present
                        if (s.find(pages[i])==s.end())
                        {
                                /// Insert current page into set
                                s.insert(pages[i]);

                                /// Increment page fault
                                page_faults++;

                                /// Push  current page into queue
                                indexes.push(pages[i]);
                        }
                }
         /// Perform FIFO if set is full
                else
                {
                        /// Check if current page is present
                        if (s.find(pages[i]) == s.end())
```

```
                        {
                                /// Store  first page in queue
                                int val = indexes.front();

                                /// Pop the first page from the queue
                                indexes.pop();

                                /// Remove index page from the set
                                s.erase(val);

                                /// insert current page in the set
                                s.insert(pages[i]);

                                /// push current page to queue
                                indexes.push(pages[i]);

                                /// Increment page faults
                                page_faults++;


                        }
                }
        }
        /// convert page faults to % of frame size
        float pfp = page_faults / n;
    cout<<n<< "            "<<frames<<"            FIFO "<<"
"<<pfp<<endl;
    //outfile
    ofstream outfile;
    outfile.open("outFile.txt", std::ios_base::app); // append instead of
overwrite
    outfile<<n<< "            "<<frames<<"            FIFO "<<"
"<<pfp<<endl;
    outfile.close();


        return 0;
}
```

lru.cpp:

```
#include <iostream>
#include<bits/stdc++.h>
#include <fstream>

using namespace std;

/// Function to find page faults using indexes
int leastRecentlyUsed(int pages[], float n, int frames)
{
    /// Declare and initialize variables
        float page_faults = 0;

        // To represent set of current pages. We use
        // an unordered_set so that we quickly check
        // if a page is present in set or not
```

```
unordered_set<int> s;

// To store least recently used indexes
// of pages.
unordered_map<int, int> indexes;


for (int i=0; i<n; i++)
{
        // Check if the set can hold more pages
        if (s.size() < frames)
        {
                // Insert it into set if not present
                // already which represents page fault
                if (s.find(pages[i])==s.end())
                {
                        s.insert(pages[i]);

                        // increment page fault
                        page_faults++;
                }

                /// Store most recently used index
                indexes[pages[i]] = i;
        }

        else
        {
                /// Check if current page is present
                if (s.find(pages[i]) == s.end())
                {
                        /// Find lru
                        int lru = INT_MAX, val;
                        /// loop to iterate through index
                        for (auto it=s.begin(); it!=s.end(); it++)
                        {
                                if (indexes[*it] < lru)
                                {
                                        lru = indexes[*it];
                                        val = *it;
                                }
                        }

                        /// Remove the index page
                        s.erase(val);

                        /// Insert the current page
                        s.insert(pages[i]);

                        /// Increment page faults
                        page_faults++;
                }

                /// Update the current page index
                indexes[pages[i]] = i;
        }
}
```

```
        ///convert page faults to % of frame size
        float pfp = page_faults / n;
        cout<< n << "            "<<frames<<"            LRU"<<  "
"<<pfp<<endl;
        /// outfile
        ofstream outfile;
        outfile.open("outFile.txt", std::ios_base::app); // append instead of
overwrite
        outfile<< n << "            "<<frames<<"            LRU"<<  "
"<<pfp<<endl;
    outfile.close();
        return 0;

}
```

mru.cpp:

```
#include <iostream>
#include<bits/stdc++.h>
#include <fstream>

using namespace std;

// Function to find page faults using indexes
int mostRecentlyUsed(int pages[], float n, int frames)
{
    /// Declare and initialize variables
        float page_faults = 0;

        // To represent set of current pages. We use
        // an unordered_set so that we quickly check
        // if a page is present in set or not
        unordered_set<int> s;

        // To store least recently used indexes
        // of pages.
        unordered_map<int, int> indexes;


        for (int i=0; i<n; i++)
        {
                // Check if the set can hold more pages
                if (s.size() < frames)
                {
                        // Insert it into set if not present
                        // already which represents page fault
                        if (s.find(pages[i])==s.end())
                        {
                                s.insert(pages[i]);

                                // increment page fault
                                page_faults++;
                        }

                        // Store the recently used index of
                        // each page
```

```
                        indexes[pages[i]] = i;
                }

                // If the set is full then need to perform mru
                // i.e. remove the least recently used page
                // and insert the current page
                else
                {
                        // Check if current page is not already
                        // present in the set
                        if (s.find(pages[i]) == s.end())
                        {
                                // Find the most recently used pages
                                // that is present in the set
                                int mru = INT_MAX, val;
                                for (auto it=s.begin(); it!=s.end(); it++)
                                {
                                        if (indexes[*it] > mru)
                                        {
                                                mru = indexes[*it];
                                                val = *it;
                                        }
                                }

                                // Remove the indexes page
                                s.erase(val);

                                // insert the current page
                                s.insert(pages[i]);

                                // Increment page faults
                                page_faults++;
                        }

                        // Update the current page index
                        indexes[pages[i]] = i;
                }
        }
        ///convert page faults to % of frame size
        float pfp = page_faults / n;

        cout<< n << "            "<<frames<<"            MRU "<<"
"<<pfp<<endl;

        //outfile
        ofstream outfile;
        outfile.open("outFile.txt", std::ios_base::app); // append instead of
overwrite
        outfile<< n << "            "<<frames<<"            MRU "<<"
"<<pfp<<endl;
        outfile.close();
        return 0;

}
```

optimal.cpp:

```
#include <bits/stdc++.h>
#include <iostream>
#include <fstream>

using namespace std;

int predict(int pages[], vector<int>& fr, float n, int index) {
   // Store the index of pagess which are going
   // to be used recently in future
   int res = -1, farthest = index;
   for (int i = 0; i < fr.size(); i++) {
      int j;
      for (j = index; j < n; j++) {
         if (fr[i] == pages[j]) {
            if (j > farthest) {
               farthest = j;
               res = i;
            }
            break;
         }
      }
      // Return the pages which are
      // are never referenced in future,
      if (j == n)
         return i;
   }
   // If all of the frames were not in future,
   // return any of them, we return 0. Otherwise
   // we return res.
   return (res == -1) ? 0 : res;
}
bool search(int key, vector<int>& fr) {
   for (int i = 0; i < fr.size(); i++)
   if (fr[i] == key)
   return true;
   return false;
}
void optimal(int pages[], float n, int frames) {
   /// Declare and initialize variables
   //float page_faults = 0;
   vector<int> fr;
   float hit = 0;
   for (int i = 0; i < n; i++) {
      // pages found in a frame : HIT
      if (search(pages[i], fr)) {
         hit++;
         continue;
      }
      //If a pages not found in a frame : MISS
      // check if there is space available in frames.
      if (fr.size() < frames)
      fr.push_back(pages[i]);
      // Find the pages to be replaced.
      else {
```

```
            int j = predict(pages, fr, n, i + 1);
            fr[j] = pages[i];
        }
    }
    //formulas to convert hits, misses and page faults to page fault %.
    float hitRatio = hit / n;
    float page_faults = n - hit;
    float pfp = page_faults / n;

    cout<< n << "            "<<frames<<"            Optimal "<<"
"<<pfp<<endl;

    // outfile
    ofstream outfile;
    outfile.open("outFile.txt", std::ios_base::app); // append instead of
overwrite
    outfile<< n << "            "<<frames<<"            Optimal "<<"
"<<pfp<<endl;
    outfile.close();
    cout<<endl;
    /*
    cout<<"***Hits and Misses***"<<endl;
    cout << "Hits = " << hit << endl;
    cout << "Misses = " << n - hit << endl;
    cout << "Hit Ratio = "<<hitRatio<<endl;
    cout << "Page Faults = "<<page_faults<<endl;
    */
}
```