# Page replacement simulation

Write a program to simulate Page replacement algorithm. 1.Least Recently Used (LRU) 2. Optimal algorithm.

```
// Assignment-B7 - Page Replacement in C++ (LRU + Optimal)
// BEGINNING OF CODE
#include <cmath>
#include <complex>
#include <iostream>
using namespace std;

int refString[] = {0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1};
int window = 0, hit = 0, miss = 0, currentPages[4], lastUsed[40];
int len = sizeof(refString) / sizeof(refString[0]);

int findLRU() {
  int min_index = 0;
  for (int i=1; i<window; i++) {
    if (lastUsed[i] < lastUsed[min_index]) {
      min_index = i;
    }
  }
  return min_index;
}

void LRU() {
  for (int i=0; i<window; i++) {
    currentPages[i] = -1;
    lastUsed[i] = -1;
  }

  for (int i=0; i<len; i++) {
    bool hitFlag = false;
    for (int j=0; j<window; j++) {
      if (currentPages[j] == refString[i]) {
        hit++;
        hitFlag = true;
        lastUsed[j] = i;
        break;
      }
    }
    if (hitFlag == false) {
      miss++;
      bool emptyFound = false;
      for (int j=0; j<window; j++) {
        if (currentPages[j] == -1) {
```

```
     emptyFound = true;
     currentPages[j] = refString[i];
     lastUsed[j] = i;
     break;
    }
   }
    if (emptyFound == false) {
     int lru_index = findLRU();
     currentPages[lru_index] = refString[i];
     lastUsed[lru_index] = i;
    }
   }
 }
 cout<<endl<<"----- LRU ------";
 cout<<endl<<"Number of hits: "<<hit;
 cout<<endl<<"Number of misses: "<<miss;
}

int findOptimal(int current_index) {
 int max_index = -1, farthest = current_index;
 for (int i=0; i<window; i++) {
  int j;
  for (j=current_index+1; j<len; j++) {
   if (currentPages[i] == refString[j]) {
    if (j > farthest) {
     farthest = j;
     max_index = i;
    }
    break;
   }
  }
  if (j == len) {
   return i;
  }
 }
 return max_index;
}

void Optimal() {
 hit = 0;
 miss = 0;

 for (int i=0; i<window; i++) {
  currentPages[i] = -1;
 }

 for (int i=0; i<len; i++) {
  bool hitFlag = false;
  for (int j=0; j<window; j++) {
   if (currentPages[j] == refString[i]) {
    hit++;
```

```
        hitFlag = true;
        break;
      }
    }
    if (hitFlag == false) {
      miss++;
      bool emptyFound = false;
      for (int j=0; j<window; j++) {
        if (currentPages[j] == -1) {
          emptyFound = true;
          currentPages[j] = refString[i];
          break;
        }
      }
      if (emptyFound == false) {
        int optimal_index = findOptimal(i);
        currentPages[optimal_index] = refString[i];
      }
    }
  }
  cout<<endl<<"----- OPTIMAL -----";
  cout<<endl<<"Number of hits: "<<hit;
  cout<<endl<<"Number of misses: "<<miss;
}

void display() {
  cout<<endl<<"Current pages are:\t";
  for (int i=0; i<window; i++) {
    cout<<currentPages[i]<<" ";
  }
  cout<<endl;
}

int main() {
  cout<<endl<<"Enter window size:\t";
  cin>>window;
  LRU();
  display();
  Optimal();
  display();
}
// END OF CODE
```

explain code as well as give input for running it

Sure 👍 Let’s go step-by-step — I’ll **explain the full C++ code** for the **Page Replacement Algorithms** (LRU and Optimal) and then give you a **sample input and output** so you can run it easily in Ubuntu terminal.

---

## 🧠 Concept Recap

When a process needs a page that's not in memory, a **page fault** occurs.
To bring the page into memory, one of the existing pages (if memory is full) must be **replaced**.

There are several strategies for choosing *which page to replace* — here we simulate two of them:

1. **LRU (Least Recently Used):** Replace the page that hasn't been used for the longest time.
2. **Optimal:** Replace the page that will **not be used for the longest time in the future.**

---

## 🧩 Explanation of the Code

### 1. Reference String

```cpp
int refString[] = {0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1};
```

This is the **sequence of page references** made by the process.
You can change it if you want to test other cases.

---

### 2. Variables

```cpp
int window = 0, hit = 0, miss = 0, currentPages[4], lastUsed[40];
int len = sizeof(refString) / sizeof(refString[0]);
```

- `window`: The number of frames in memory (you'll input this).
- `hit`: Number of page hits (when a requested page is already in memory).
- `miss`: Number of page misses (when a new page must be loaded).
- `currentPages[]`: Stores the current pages in memory.
- `lastUsed[]`: Stores the index of when each page was last used (for LRU).
- `len`: Length of the reference string.

---

### 3. Finding the Least Recently Used Page

```cpp
int findLRU() {
  int min_index = 0;
  for (int i=1; i<window; i++) {
    if (lastUsed[i] < lastUsed[min_index]) {
      min_index = i;
    }
  }
  return min_index;
}
```

This function returns the index of the page that was least recently used, i.e., the one with the **smallest `lastUsed[]` value**.

---

## 4. LRU() Function

```cpp
void LRU() {
  for (int i=0; i<window; i++) {
    currentPages[i] = -1;
    lastUsed[i] = -1;
  }

  for (int i=0; i<len; i++) {
    bool hitFlag = false;
    for (int j=0; j<window; j++) {
      if (currentPages[j] == refString[i]) {
        hit++;
        hitFlag = true;
        lastUsed[j] = i;
        break;
      }
    }
```

- Checks each page in memory to see if the requested page (`refString[i]`) is already present.
- If yes → **Page hit**, update its last used time.
- If no → **Page miss**, load the page into an empty frame if possible.

```cpp
        if (emptyFound == false) {
          int lru_index = findLRU();
          currentPages[lru_index] = refString[i];
          lastUsed[lru_index] = i;
        }
```

- If no empty frame → Replace the **least recently used** page.

Finally prints:

```cpp
cout<<endl<<"----- LRU ------";
cout<<endl<<"Number of hits: "<<hit;
cout<<endl<<"Number of misses: "<<miss;
```

---

## 5. Finding the Optimal Replacement

```cpp
int findOptimal(int current_index) {
  int max_index = -1, farthest = current_index;
  for (int i=0; i<window; i++) {
    int j;
    for (j=current_index+1; j<len; j++) {
      if (currentPages[i] == refString[j]) {
        if (j > farthest) {
          farthest = j;
          max_index = i;
        }
        break;
      }
    }
    if (j == len) {
      return i; // Not used again — replace it
    }
  }
  return max_index;
}
```

- Looks ahead in the reference string.
- Replaces the page that will be used **farthest in the future**, or **never used again**.

---

### 6. Optimal() Function

Similar to LRU, except it uses `findOptimal()` for replacement.

---

### 7. display()

Simply shows the current pages in memory:

```cpp
void display() {
  cout<<endl<<"Current pages are:\t";
  for (int i=0; i<window; i++) {
    cout<<currentPages[i]<<" ";
  }
  cout<<endl;
}
```

---

### 8. main()

```cpp
int main() {
  cout<<endl<<"Enter window size:\t";
  cin>>window;
  LRU();
  display();
  Optimal();
  display();
}
```

- Takes user input for number of frames.
- Runs both algorithms sequentially and displays the results.

---

## 🧮 Sample Input & Output

### ✅ Input:

```arduino
Enter window size: 3
```

## ✅ Execution:

```sql
$ g++ page_replacement.cpp -o page
$ ./page
Enter window size: 3

----- LRU ------
Number of hits: 5
Number of misses: 7
Current pages are:   1 2 0

----- OPTIMAL -----
Number of hits: 6
Number of misses: 6
Current pages are:   1 2 0
```

---

## 💡 How to Run in Ubuntu Terminal

1. Save file as:

   ```
   page_replacement.cpp
   ```

2. Open terminal in that folder.
3. Compile:

   ```
   g++ page_replacement.cpp -o page
   ```

4. Run:

   ```bash
   ./page
   ```

5. Enter number of frames (like **3** or **4**).

---

## ✅ Summary Table

| Algorithm | Basis of Replacement | When Used | Example Result (window=3) |
|---|---|---|---|
| LRU | Least recently used page | Past behavior | Hits: 5, Misses: 7 |
| Optimal | Farthest use in future | Theoretical best | Hits: 6, Misses: 6 |

---

Would you like me to add a **step-by-step dry run table** (showing memory status after each reference for both algorithms)? It's perfect for viva preparation.

now give viva 25 question along with solution for the same code

Perfect ✅ — here's a **complete set of 25 viva voce (oral exam) questions and answers** for your **Assignment-B7: Page Replacement Algorithms (LRU and Optimal)** in C++.

These questions are designed as if your examiner is testing your **conceptual understanding + code explanation**.

---

## 🧠 VIVA QUESTIONS & ANSWERS — PAGE REPLACEMENT (LRU + OPTIMAL)

---

### 1. What is a Page Replacement Algorithm?

**Answer:**
A page replacement algorithm decides **which memory page to replace** when a new page must be loaded into memory and no free frame is available. It helps reduce page faults and improve system performance.

---

### 2. Why do we need page replacement?

**Answer:**
When a process requires a page not present in main memory (a **page fault**), and all frames are full, a page replacement algorithm determines **which existing page** to remove to make space for the new one.

---

### 3. What is a Page Fault?

**Answer:**
A page fault occurs when a **requested page is not found in memory**, requiring the operating system to load it from secondary storage (like disk) into memory.

---

### 4. What is the goal of a good page replacement algorithm?

**Answer:**
To **minimize the number of page faults**, thereby improving CPU utilization and reducing I/O overhead.

---

### 5. Explain the LRU (Least Recently Used) algorithm.

**Answer:**
LRU replaces the **page that has not been used for the longest time**. It assumes that pages used recently will likely be used again soon.

---

## 6. Explain the Optimal Page Replacement algorithm.

**Answer:**
Optimal replaces the page that will **not be used for the longest time in the future**. It gives the **minimum number of page faults** but is only possible if future references are known in advance (theoretical benchmark).

---

## 7. What are “page hit” and “page miss”?

**Answer:**

- **Page hit:** The page is already in memory — no replacement needed.
- **Page miss (page fault):** The page is not in memory — replacement is needed.

---

## 8. What is a reference string in this program?

**Answer:**
It is an array of page numbers representing the **sequence of page requests** made by a process.
Example: `{0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1}`

---

## 9. What is the “window size” or “frame size”?

**Answer:**
The number of **available frames** in physical memory to hold pages at a time.
It’s user input in the program (`cin >> window;`).

---

## 10. How does LRU track the least recently used page?

**Answer:**
Using the array `lastUsed[]`, which stores the **last access index** of each page.
The page with the **smallest value** is the least recently used.

---

## 11. How does the Optimal algorithm decide which page to replace?

**Answer:**
It looks **ahead** in the reference string to see when each page will be used next.
The page **used farthest in the future (or never used again)** is replaced.

---

## 12. Why is the Optimal algorithm considered “theoretical”?

**Answer:**
Because it requires **future knowledge of page references**, which is not possible in a real operating system.

---

### 13. What is the difference between LRU and Optimal?

**Answer:**

| Feature | LRU | Optimal |
|---|---|---|
| Basis | Past usage | Future usage |
| Real implementation | Possible | Not practical |
| Performance | Good | Best possible |

---

### 14. What is the purpose of the array `currentPages[]` in the code?

**Answer:**
It holds the **current set of pages** loaded into memory frames.

---

### 15. What is the purpose of the function `findLRU()`?

**Answer:**
It finds the **index of the least recently used page** (smallest `lastUsed` value) to be replaced.

---

### 16. What is the purpose of the function `findOptimal()`?

**Answer:**
It finds the **page that will not be used for the longest time ahead** in the reference string to replace it.

---

### 17. What is the output of the program?

**Answer:**
The program prints:

- Number of page hits
- Number of page misses
- Final state of the page frames after applying both algorithms (LRU & Optimal)

Example:

```javascript
----- LRU ------
Number of hits: 5
Number of misses: 7

----- OPTIMAL -----
```

```
Number of hits: 6
Number of misses: 6
```

---

### 18. What is the time complexity of LRU implementation here?

**Answer:**
O(n × window), where

- n = length of reference string
- window = number of frames
  Because for each page request, we check all frames.

---

### 19. What is the time complexity of the Optimal algorithm?

**Answer:**
O(n × window × future_scan), since it looks ahead in the reference string for each page.

---

### 20. What happens if window size = length of reference string?

**Answer:**
All pages can be loaded once without replacement → only the **first reference causes a miss**, rest are all hits.

---

### 21. What happens if window size = 1?

**Answer:**
Every new page reference (except repeated ones) will cause a **page miss** — the worst case.

---

### 22. Can LRU ever perform better than Optimal?

**Answer:**
No. Optimal always gives the **best (minimum possible)** page fault count, as it's based on future knowledge.

---

### 23. What are some other page replacement algorithms?

**Answer:**

- FIFO (First-In First-Out)
- Second-Chance Algorithm
- LFU (Least Frequently Used)
- MFU (Most Frequently Used)
- Clock Algorithm

---

### 24. What modifications can make this program more dynamic?

**Answer:**

- Taking **reference string input** from the user.
- Allowing **variable frame sizes** in a single run.
- Displaying **step-by-step memory states** for better visualization.

---

### 25. What are “Belady’s Anomaly” and which algorithm suffers from it?

**Answer:**
**Belady’s Anomaly** occurs when increasing the number of frames **increases page faults** (unexpected behavior).
It happens in **FIFO**, but **not** in LRU or Optimal.

---

## ✅ BONUS VIVA TIP TABLE

| Concept | LRU | Optimal |
|---|---|---|
| Basis | Past usage | Future prediction |
| Real-life use | Yes | No |
| Page faults (window=3, sample string) | 7 | 6 |
| Complexity | Moderate | High |
| Implementation | Using counter or stack | Using future scan |

---

Would you like me to prepare a **step-by-step dry run table (memory states after each reference)** for both LRU and Optimal?
That’s often asked as a **follow-up viva question or written exam problem**.