1. FIFO Page Replacement Algorithm

Aim

To implement FIFO Page Replacement Algorithm in C and calculate the number of page faults.

Algorithm (with variables)

1. Start the program.
2. Read **frames** (number of frames) and **pages** (number of pages).
3. Read reference string into array **ref[]**.
4. Initialize **frame[]** with `-1`.
5. Set **pageFaults = 0**, **k = 0**.
6. For each page in **ref[]**:
   a) If page exists in **frame[]**, continue.
   b) Else replace **frame[k]** with the page and increment **pageFaults**.
   c) Update **k = (k + 1) % frames**.
7. Print frames after each step and total page faults.
8. Stop.

C Program

```
#include <stdio.h>
int main() {
   int frames, pages, i, j, k = 0, pageFaults = 0;
   printf("Enter number of frames: ");
   scanf("%d", &frames);
   printf("Enter number of pages: ");
   scanf("%d", &pages);
   int ref[pages], frame[frames];
   printf("Enter reference string: ");
   for (i = 0; i < pages; i++) scanf("%d", &ref[i]);
```

```
    for (i = 0; i < frames; i++) frame[i] = -1;
    printf("\nPage\tFrames\n");
    for (i = 0; i < pages; i++) {
        int found = 0;
        for (j = 0; j < frames; j++) if (frame[j] == ref[i]) found = 1;
        if (!found) {
            frame[k] = ref[i];
            k = (k + 1) % frames;
            pageFaults++;
        }
        printf("%d\t", ref[i]);
        for (j = 0; j < frames; j++) frame[j] == -1 ? printf("- ") : printf("%d ", frame[j]);
        printf("\n");
    }
    printf("\nTotal Page Faults: %d\n", pageFaults);
}
```

Step-by-Step Explanation

- Read input for **frames**, **pages**, and reference string.
- Initialize all frames to `-1` (empty).
- For each page:
    - If present in frame → no page fault.
    - Else → replace using **FIFO order** (oldest page), update **k** circularly.
- Print frame contents and total page faults.

2. LRU Page Replacement Algorithm

Aim

To implement LRU Page Replacement Algorithm in C and calculate page faults.

Algorithm (with variables)

1. Read **frames** and **pages**.

2. Read reference string into **ref[]**.

3. Initialize **frame[] = -1**, **recent[] = 0**, **time = 0**, **pageFaults = 0**.

4. For each page in **ref[]**:
   a) If page exists in **frame[]**, update its recent time.
   b) Else if empty frame exists, insert page and update time.
   c) Else find page with **smallest recent[]** (least recently used) and replace.
   d) Increment **pageFaults** and **time**.

5. Print frames after each step and total page faults.

C Program

```
#include <stdio.h>
int main() {
   int frames, pages;
   printf("Enter number of frames: ");
   scanf("%d", &frames);
   printf("Enter number of pages: ");
   scanf("%d", &pages);
   int ref[pages], frame[frames], recent[frames];
   printf("Enter reference string: ");
   for (int i = 0; i < pages; i++) scanf("%d", &ref[i]);
   for (int i = 0; i < frames; i++) frame[i] = -1, recent[i] = 0;
   int time = 0, pageFaults = 0;
   printf("\nPage\tFrames\n");
   for (int i = 0; i < pages; i++) {
      int page = ref[i], found = 0;
      for (int j = 0; j < frames; j++) {
```

```
            if (frame[j] == page) { found = 1; recent[j] = ++time; break; }
        }
        if (!found) {
            int emptyPos = -1;
            for (int j = 0; j < frames; j++) if (frame[j] == -1) { emptyPos = j; break; }
            if (emptyPos != -1) frame[emptyPos] = page, recent[emptyPos] = ++time;
            else {
                int lruPos = 0;
                for (int j = 1; j < frames; j++) if (recent[j] < recent[lruPos]) lruPos = j;
                frame[lruPos] = page; recent[lruPos] = ++time;
            }
            pageFaults++;
        }
        printf("%d\t", page);
        for (int j = 0; j < frames; j++) frame[j] == -1 ? printf("- ") : printf("%d ", frame[j]);
        printf("\n");
    }
    printf("\nTotal Page Faults: %d\n", pageFaults);
}
```

Step-by-Step Explanation

1. **Input:**
    - Number of frames (**frames**)
    - Number of pages (**pages**)
    - Reference string (**ref[]**)
2. **Initialization:**
    - `frame[] = -1` (to indicate empty slots)
    - `recent[] = 0` (stores last used time for each frame)
    - `time = 0`, `pageFaults = 0`
3. **Process each page in ref[]:**

- **Case 1: Page is already in a frame**
  → No page fault, just update `recent[frame_position] = ++time`

- **Case 2: Page is NOT in frames**

  - If there is an **empty frame**
    → Insert page into that frame and update `recent`

  - Else (all frames full)
    → Find **frame with smallest recent[] value** (Least Recently Used)
    → Replace that page with the current page

4. **Increment `pageFaults` whenever a new page is inserted/replaced**.

5. **Print** current frames after each page reference.

6. **After all pages processed**, display total page faults.

3. Optimal Page Replacement Algorithm

Aim

To implement Optimal Page Replacement Algorithm in C and calculate page faults.

Algorithm

1. Read **frames**, **pages**, and reference string.

2. Initialize **frame[] = -1**, **pageFaults = 0**.

3. For each page:
   a) If present, continue.
   b) Else if empty frame, insert page.
   c) Else replace page which is used farthest in future.

4. Print frames after each step and total page faults.

C Program

```
#include <stdio.h>
int predict(int ref[], int frame[], int pages, int index, int frames) {
    int farthest = index, pos = -1;
    for (int i = 0; i < frames; i++) {
        int j;
        for (j = index; j < pages; j++) {
            if (frame[i] == ref[j]) {
                if (j > farthest) { farthest = j; pos = i; }
                break;
            }
        }
        if (j == pages) return i;
    }
    return (pos == -1) ? 0 : pos;
}
int main() {
    int frames, pages;
    printf("Enter number of frames: ");
```

```
    scanf("%d", &frames);
    printf("Enter number of pages: ");
    scanf("%d", &pages);
    int ref[pages], frame[frames];
    printf("Enter reference string: ");
    for (int i = 0; i < pages; i++) scanf("%d", &ref[i]);
    for (int i = 0; i < frames; i++) frame[i] = -1;
    int pageFaults = 0;
    printf("\nPage\tFrames\n");
    for (int i = 0; i < pages; i++) {
        int page = ref[i], found = 0;
        for (int j = 0; j < frames; j++) if (frame[j] == page) { found = 1; break; }
        if (!found) {
            int emptyPos = -1;
            for (int j = 0; j < frames; j++) if (frame[j] == -1) { emptyPos = j; break; }
            if (emptyPos != -1) frame[emptyPos] = page;
            else { int pos = predict(ref, frame, pages, i+1, frames); frame[pos] = page; }
            pageFaults++;
        }
        printf("%d\t", page);
        for (int j = 0; j < frames; j++) frame[j] == -1 ? printf("- ") : printf("%d ", frame[j]);
        printf("\n");
    }
    printf("\nTotal Page Faults: %d\n", pageFaults);
}
```

Step-by-Step Explanation

1. **Input:**
    - Number of frames (**frames**)
    - Number of pages (**pages**)
    - Reference string (**ref[]**)
2. **Initialization:**
    - `frame[] = -1` (empty frames)
    - `pageFaults = 0`

3. **Process each page in ref[]:**
    - **Case 1: Page already exists in frames**
      → Do nothing
    - **Case 2: Page not in frames**
        - If **empty frame available**
          → Insert page into empty frame
        - Else (frames full)
          → Predict which page will **not be used for the longest time in the future**
          → Replace that page (function `predict()` finds this)
4. **Increment `pageFaults` whenever a replacement occurs**.
5. **Print** current frame content after every step.
6. At the end, display total page faults.