UCS 1411 - Operating Systems Lab
Exercise 10 – Page Replacement Technique

Mahesh Bharadwaj K - 185001089

Develop a C program to implement the page replacement algorithms (FIFO, Optimal, LRU and LFU) using linked list

Linked List Header File

```
typedef int Data;

typedef struct Node
{
    Data d;
    struct Node *next;
    int freq;
} Node;

typedef Node *List;

List createEmptyList()
{
    Node *head = (Node *)malloc(sizeof(Node));
    head->d = 0;
    head->next = NULL;
    return head;
}

void insertLast(List head, const Data d)
{
    Node *new = (Node *)malloc(sizeof(Node));
    new->d = d;
    new->freq = 1;
    Node *tmp = head;

    while (tmp->next != NULL)
        tmp = tmp->next;

    new->next = NULL;
    tmp->next = new;
}

void insertFirst(List head, const Data d)
{
    Node *new = (Node *)malloc(sizeof(Node));
    new->d = d;
    new->freq = 1;

    new->next = head->next;
    head->next = new;
}
```

```c
Data delete (List prev)
{
    Data rVal = -1;
    if (!prev)
        return rVal;
    if (!prev->next)
        return rVal;

    Node *tmp = prev->next;
    rVal = tmp->d;
    prev->next = prev->next->next;
    free(tmp);

    return rVal;
}

Data deleteFirst(List head)
{
    Data rVal = -1;
    if (head->next == NULL)
    {
        printf(" Empty List!\n");
        return rVal;
    }

    delete (head);
}

Data deleteLast(List head)
{
    Data rVal = -1;
    if (head->next == NULL)
    {
        printf(" Empty List!\n");
        return rVal;
    }

    Node *tmp = head;
    while (tmp->next->next != NULL)
        tmp = tmp->next;

    delete (tmp);
}

void display(List head)
{
    Node *tmp = head->next;

    if (tmp == NULL)
    {
        printf(" Empty!\n");
        return;
    }

    while (tmp)
    {
        printf(" %-2d", tmp->d);
        tmp = tmp->next;
    }
}

int length(List head)
{
    Node *tmp = head->next;
    if (tmp == NULL)
        return 0;

    int count = 0;
    while (tmp)
    {
        tmp = tmp->next;
        count++;
    }
    return count;
}

List search(List head, const Data d)
{
    if (head->next == NULL)
        return NULL;

    Node *tmp = head;
    while (tmp->next)
    {
        if (tmp->next->d == d)
            return tmp;
        tmp = tmp->next;
    }

    return NULL;
}
```

Main Program

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#include "LinkedList.h"

#define ROW 10
#define COL 20

int *const convert(const char *const, int *);

void FIFO(const int *const, const int, const int);
void optimal(const int *const, const int, const int);
void LRU(const int *const, const int, const int);
void LFU(const int *const, const int, const int);
void putTable(const int[ROW][COL], const int, const int);

int main()
{
    int n_free_frames = -1;
    int n_reqd_frames = -1;
    char buffer[20] = {0};
    int *sequence = NULL;
    int choice = -1;
    int len = 0;

    while (1)
    {
        printf("\t\t\t\tPAGE REPLACEMENT TECHNIQUES\n");
        printf(" 1 - Read Input\n");
        printf(" 2 - FIFO\n");
        printf(" 3 - Optimal\n");
        printf(" 4 - LRU\n");
        printf(" 5 - LFU\n");
        printf(" 0 - Exit\n");
        printf(" -------------------------\n");
        printf(" Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 0:
            exit(0);
        case 1:
            printf(" Enter the number of free frames: ");
            scanf("%d", &n_free_frames);
            printf(" Enter the number of required frames: ");
            scanf("%d", &n_reqd_frames);
            getchar();
            printf(" Enter the Reference String: ");
            scanf("%[^\n]", buffer);
            sequence = convert(buffer, &len);
            break;
        case 2:
            printf("\n\t\tFIFO\n");
            FIFO(sequence, len, n_reqd_frames);
            break;
        case 3:
            printf("\n\t\t\tOPTIMAL\n");
            optimal(sequence, len, n_reqd_frames);
            break;
        case 4:
            printf("\n\t\tLRU\n");
            LRU(sequence, len, n_reqd_frames);
            break;
        case 5:
            printf("\n\t\tLFU\n");
            LFU(sequence, len, n_reqd_frames);
            break;
        default:
            printf(" Invalid Input!\n");
        }
        printf("\n");
    }
}

int *const convert(const char *const refstr, int *size)
```

```
{
    static int arr[30];
    int i = 0, val = 0;

    while (refstr[i])
    {
        if (isdigit(refstr[i]))
        {
            val = refstr[i] - 48;
            for (int j = i + 1; refstr[j] && isdigit(refstr[j]); j++)
            {
                val = (val * 10) + (refstr[j] - 48);
                i = j;
            }

            arr[*size] = val;
            (*size)++;
        }
        i++;
    }
    return arr;
}

void putTable(const int table[ROW][COL], const int n_frames, const int n_updates)
{
    printf("\n ");
    for (int i = 0; i < n_updates; i++)
        printf("+----");
    printf("+\n ");

    for (int i = 0; i < n_frames; i++)
    {
        for (int j = 0; j < n_updates; j++)
        {
            if (table[i][j] == -1)
                printf("| -  ");
            else
                printf("| %-2d ", table[i][j]);
        }
        printf("|\n ");
    }
    for (int i = 0; i < n_updates; i++)
        printf("+----");
    printf("+\n ");
}

void insertTable(List tmp, int table[ROW][COL], const int n_frames, const int faults)
{
    for (int i = 0; i < n_frames; i++)
    {
        if (tmp)
        {
            table[i][faults] = tmp->d;
            tmp = tmp->next;
        }
        else
            table[i][faults] = -1;
    }
}

void FIFO(const int *const seq, const int len, const int n_frames)
{
    int size = 0;
    int faults = 0;
    int table[ROW][COL];

    List alloc = createEmptyList();

    Node *oldest;

    printf("\n");
    printf("   Frame ->        In Memory      -> Faults \n\n");

    for (int i = 0; i < len; i++)
    {
        printf("     %-2d  ->", seq[i]);

        Node *isFound = search(alloc, seq[i]);
        Node *tmp;

        if (!isFound)
        {
            if (size < n_frames)
            {
                insertLast(alloc, seq[i]);
                size++;

                //Initialise first frame as oldest
                if (size == 1)
                    oldest = alloc->next;
            }
            else
            {

                //Swap oldest frame with new frame
                oldest->d = seq[i];

                //Update oldest frame
                if (oldest->next)
                    oldest = oldest->next;
                else
                    oldest = alloc->next;
            }
            //Updating Table
            insertTable(alloc -> next, table, n_frames, faults);
            faults++;
        }
        display(alloc);
        for (int i = length(alloc) * 3; i <= 22; i++)
            printf(" ");
        printf("->    %-2d   \n", faults);
    }
    putTable(table, n_frames, faults);
}

void optimal(const int *const seq, const int len, const int n_frames)
```

```
{
    int size = 0;
    int faults = 0;
    int distance;
    int flag;
    int table[ROW][COL];

    List alloc = createEmptyList();

    Node *farthest = NULL, *tmp;

    printf("\n");
    printf("   Frame ->        In Memory      -> Faults \n\n");

    int val = 0;
    int i = 0;
    for (int i = 0; i < len; i++)
    {
        printf("     %-2d  ->", seq[i]);

        Node *isFound = search(alloc, seq[i]);

        if (!isFound)
        {
            if (size < n_frames)
            {
                insertLast(alloc, seq[i]);
                size++;
            }
            else
            {
                tmp = alloc->next;
                distance = 0;

                //Find the frame which is used the farthest away and swap
                while (tmp)
                {
                    flag = 0;

                    for (int j = i + 1; j < len; j++)
                    {
                        if (seq[j] == tmp->d)
                        {
                            flag = 1;
                            if (j - i > distance)
                            {
                                distance = (j - i);
                                farthest = tmp;
                            }
                            break;
                        }
                    }

                    //Not Used in the future
                    if (!flag)
                    {
                        farthest = tmp;
                        break;
```

```c
                    }
                    tmp = tmp->next;
                }

                farthest->d = seq[i];
            }
            //Updating Table
            insertTable(alloc -> next, table, n_frames, faults);
            faults++;

        }
        display(alloc);
        for (int i = length(alloc) * 3; i <= 22; i++)
            printf(" ");
        printf("->   %-2d   \n", faults);
    }
    putTable(table, n_frames, faults);

}

void LRU(const int *const seq, const int len, const int n_frames)
{
    int size = 0;
    int faults = 0;
    int distance;
    int table[ROW][COL];

    List alloc = createEmptyList();

    Node *least_recent = NULL, *tmp;

    printf("\n");
    printf("   Frame ->        In Memory      -> Faults \n\n");

    int val = 0;
    int i = 0;
    for (int i = 0; i < len; i++)
    {
        printf("     %-2d  ->", seq[i]);

        Node *isFound = search(alloc, seq[i]);

        if (!isFound)
        {
            if (size < n_frames)
            {
                insertLast(alloc, seq[i]);
                size++;
            }
            else
            {
                tmp = alloc->next;
                distance = 0;

                //Find the frame which is used the least recently and swap
                while (tmp)
                {
                    for (int j = i - 1; j >= 0; j--)
                    {
                        if (seq[j] == tmp->d)
                        {
                            if (i - j > distance)
                            {
                                distance = (i - j);
                                least_recent = tmp;
                            }
                            break;
                        }
                    }
                    tmp = tmp->next;
                }
                least_recent->d = seq[i];
            }
            //Updating Table
            insertTable(alloc -> next, table, n_frames, faults);
            faults++;

        }
        display(alloc);
        for (int i = length(alloc) * 3; i <= 22; i++)
            printf(" ");
        printf("->   %-2d   \n", faults);
    }
    putTable(table, n_frames, faults);

}

void LFU(const int *const seq, const int len, const int n_frames)
{
    int size = 0;
    int faults = 0;
    int frequency;
    int table[ROW][COL];

    List alloc = createEmptyList();

    Node *least_frequent = NULL, *tmp;

    printf("\n");
    printf("   Frame ->        In Memory      -> Faults \n\n");

    int val = 0;
    int i = 0;
    for (int i = 0; i < len; i++)
    {
        printf("     %-2d  ->", seq[i]);

        Node *isFound = search(alloc, seq[i]);

        if (!isFound)
        {

            if (size < n_frames)
            {
                insertLast(alloc, seq[i]);
                size++;
```

```c
            }
            else
            {
                tmp = alloc->next;
                frequency = 99;

                //Find the frame which is least frequently used and swap
                while (tmp)
                {
                    if (tmp->freq < frequency)
                    {
                        frequency = tmp->freq;
                        least_frequent = tmp;
                    }
                    tmp = tmp->next;
                }
                least_frequent->d = seq[i];
                least_frequent->freq = 1;
            }
            //Updating Table
            insertTable(alloc -> next, table, n_frames, faults);
            faults++;

        }
        else
            isFound->next->freq++;
        display(alloc);
        for (int i = length(alloc) * 3; i <= 22; i++)
            printf(" ");
        printf("->   %-2d   \n", faults);
    }
    putTable(table, n_frames, faults);

}
```

Output

```
PAGE REPLACEMENT TECHNIQUES
 1 - Read Input
 2 - FIFO
 3 - Optimal
 4 - LRU
 5 - LFU
 0 - Exit
 -------------------------
 Enter your choice: 1
 Enter the number of free frames: 10
 Enter the number of required frames: 3
 Enter the Reference String: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

PAGE REPLACEMENT TECHNIQUES
 1 - Read Input
 2 - FIFO
 3 - Optimal
 4 - LRU
 5 - LFU
 0 - Exit
 -------------------------
 Enter your choice: 2

FIFO

   Frame ->        In Memory      -> Faults

     7   -> 7                     ->    1
     0   -> 7  0                  ->    2
     1   -> 7  0  1               ->    3
     2   -> 2  0  1               ->    4
     0   -> 2  0  1               ->    4
     3   -> 2  3  1               ->    5
     0   -> 2  3  0               ->    6
     4   -> 4  3  0               ->    7
     2   -> 4  2  0               ->    8
     3   -> 4  2  3               ->    9
     0   -> 0  2  3               ->    10
     3   -> 0  2  3               ->    10
     2   -> 0  2  3               ->    10
     1   -> 0  1  3               ->    11
     2   -> 0  1  2               ->    12
     0   -> 0  1  2               ->    12
     1   -> 0  1  2               ->    12
     7   -> 7  1  2               ->    13
     0   -> 7  0  2               ->    14
     1   -> 7  0  1               ->    15

 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 | 7  | 7  | 7  | 2  | 2  | 2  | 4  | 4  | 4  | 0  | 0  | 0  | 7  | 7  | 7  |
 | -  | 0  | 0  | 0  | 3  | 3  | 3  | 2  | 2  | 2  | 1  | 1  | 1  | 0  | 0  |
 | -  | -  | 1  | 1  | 1  | 0  | 0  | 0  | 3  | 3  | 3  | 2  | 2  | 2  | 1  |
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+

PAGE REPLACEMENT TECHNIQUES
 1 - Read Input
 2 - FIFO
 3 - Optimal
 4 - LRU
 5 - LFU
 0 - Exit
 -------------------------
 Enter your choice: 3

OPTIMAL

   Frame ->        In Memory      -> Faults

     7   -> 7                     ->   1
     0   -> 7  0                  ->   2
     1   -> 7  0  1               ->   3
     2   -> 2  0  1               ->   4
     0   -> 2  0  1               ->   4
     3   -> 2  0  3               ->   5
     0   -> 2  0  3               ->   5
     4   -> 2  4  3               ->   6
     2   -> 2  4  3               ->   6
     3   -> 2  4  3               ->   6
```

```
    0   -> 2  0  3              ->   7
    3   -> 2  0  3              ->   7
    2   -> 2  0  3              ->   7
    1   -> 2  0  1              ->   8
    2   -> 2  0  1              ->   8
    0   -> 2  0  1              ->   8
    1   -> 2  0  1              ->   8
    7   -> 7  0  1              ->   9
    0   -> 7  0  1              ->   9
    1   -> 7  0  1              ->   9

 +----+----+----+----+----+----+----+----+----+
 | 7  | 7  | 7  | 2  | 2  | 2  | 2  | 2  | 7  |
 | -  | 0  | 0  | 0  | 0  | 4  | 0  | 0  | 0  |
 | -  | -  | 1  | 1  | 3  | 3  | 3  | 1  | 1  |
 +----+----+----+----+----+----+----+----+----+

PAGE REPLACEMENT TECHNIQUES
 1 - Read Input
 2 - FIFO
 3 - Optimal
 4 - LRU
 5 - LFU
 0 - Exit
 -------------------------
 Enter your choice: 4

LRU

   Frame ->        In Memory      -> Faults

    7   -> 7                    ->   1
    0   -> 7  0                 ->   2
    1   -> 7  0  1              ->   3
    2   -> 2  0  1              ->   4
    0   -> 2  0  1              ->   4
    3   -> 2  0  3              ->   5
    0   -> 2  0  3              ->   5
    4   -> 4  0  3              ->   6
    2   -> 4  0  2              ->   7
    3   -> 4  3  2              ->   8
    0   -> 0  3  2              ->   9
    3   -> 0  3  2              ->   9
    2   -> 0  3  2              ->   9
    1   -> 1  3  2              ->   10
    2   -> 1  3  2              ->   10
    0   -> 1  0  2              ->   11
    1   -> 1  0  2              ->   11
    7   -> 1  0  7              ->   12
    0   -> 1  0  7              ->   12
    1   -> 1  0  7              ->   12

 +----+----+----+----+----+----+----+----+----+----+----+----+
 | 7  | 7  | 7  | 2  | 2  | 4  | 4  | 4  | 0  | 1  | 1  | 1  |
 | -  | 0  | 0  | 0  | 0  | 0  | 0  | 3  | 3  | 3  | 0  | 0  |
 | -  | -  | 1  | 1  | 3  | 3  | 2  | 2  | 2  | 2  | 2  | 7  |
 +----+----+----+----+----+----+----+----+----+----+----+----+
PAGE REPLACEMENT TECHNIQUES
 1 - Read Input
 2 - FIFO
 3 - Optimal
 4 - LRU
 5 - LFU
 0 - Exit
 -------------------------
 Enter your choice: 5

LFU

   Frame ->        In Memory      -> Faults

     7   -> 7                     ->   1
     0   -> 7  0                  ->   2
     1   -> 7  0  1               ->   3
     2   -> 2  0  1               ->   4
     0   -> 2  0  1               ->   4
     3   -> 3  0  1               ->   5
     0   -> 3  0  1               ->   5
     4   -> 4  0  1               ->   6
     2   -> 2  0  1               ->   7
     3   -> 3  0  1               ->   8
     0   -> 3  0  1               ->   8
     3   -> 3  0  1               ->   8
     2   -> 3  0  2               ->   9
     1   -> 3  0  1               ->   10
     2   -> 3  0  2               ->   11
     0   -> 3  0  2               ->   11
     1   -> 3  0  1               ->   12
     7   -> 3  0  7               ->   13
     0   -> 3  0  7               ->   13
     1   -> 3  0  1               ->   14

 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 | 7  | 7  | 7  | 2  | 3  | 4  | 2  | 3  | 3  | 3  | 3  | 3  | 3  | 3  |
 | -  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
 | -  | -  | 1  | 1  | 1  | 1  | 1  | 1  | 2  | 1  | 2  | 1  | 7  | 1  |
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+

PAGE REPLACEMENT TECHNIQUES
 1 - Read Input
 2 - FIFO
 3 - Optimal
 4 - LRU
 5 - LFU
 0 - Exit
 -------------------------
 Enter your choice: 0
```