Department of Computer Science and Engineering

S.G.Shivanirudh , 185001146, Semester IV

28 March 2020

UCS1411 - Operating Systems Laboratory

Lab Exercise 10: Implementation of Page Replacement Algorithms

Objective:

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

Code:

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

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<string.h>
4
5 #define hole "H"
```

```
6
7  struct Box{
8      int page_no;
9      struct Box* next;
10 };
11
12 typedef struct Box Frame;
13
14 void initialise(Frame* f){
15
16     f->page_no=-1;
17     f->next=NULL;
18 }
19
20 //Copy one partition to another
21 void copyFrame(Frame *copy,Frame *OG){
22     copy->page_no=OG->page_no;
23     copy->next=NULL;
24 }
25
26 //Count number of partitions
27 int partitionCount(Frame* LL){
28     Frame *tmp=LL;
29     int ctr=0;
30     tmp=tmp->next;
31     while(tmp){
32         ctr++;
33         tmp=tmp->next;
34     }
35     return ctr;
36 }
37
38 //Insert into Linked List
39 void insert(Frame *LL, Frame *n){
40
41     Frame *stmp=LL;
42     while(stmp->next){
43         stmp=stmp->next;
44     }
45     n->next=stmp->next;
46     stmp->next=n;
47 }
48
49 //Check if an element is present in the list
50 int checkLL(Frame *LL,int x){
```

```
51      Frame *tmp=LL;
52      tmp=tmp->next;
53      while(tmp){
54          if(x==tmp->page_no)
55              return 1;
56          tmp=tmp->next;
57      }
58      return 0;
59  }
60
61  //Delete first element of linked list
62  void deleteFirst(Frame *LL){
63      Frame *tmp=LL;
64      Frame *prev=tmp;
65      tmp=tmp->next;
66      if(tmp->next){
67          prev->next=tmp->next;
68          tmp->next=NULL;
69          free(tmp);
70      }
71      else{
72          printf("\n Empty\n");
73      }
74  }
75
76
77  //Display Frames
78  void displayFrames(Frame *LL,int frames_reqd){
79
80      int no_of_partitions=partitionCount(LL);
81
82      int spacing = 27/frames_reqd;
83
84      Frame *tmp=LL;
85      tmp=tmp->next;
86
87      for(int i=0;i<frames_reqd;i++){
88          if(tmp==NULL){
89              printf("%-5c",'-');
90              continue;
91          }
92          else{
93              printf("%-5d",tmp->page_no);
94              tmp=tmp->next;
95          }
```

```
    }
    for(int i=0;i<27-(5*frames_reqd);i++)
        printf(" ");
}

//Replace a page with another
void ReplacePos(Frame *LL,int old_page_no,int new_page_no){
    Frame *tmp=LL;

    tmp=tmp->next;

    while(tmp){
        if(tmp->page_no!=old_page_no){
            tmp=tmp->next;
        }
        else
            break;
    }
    tmp->page_no=new_page_no;
}
```

```
#include "LinkedList.h"

//Accept data
void acceptData(int *noff,int *frames_reqd,char *
    reference_string){
    printf("\nEnter the number of free frames: ");scanf("%d",
    noff);

    do{
        printf("\nEnter the number of frames required: ");
        scanf("%d",frames_reqd);
        if(*frames_reqd>*noff)
            printf("\nProcess requires more frames than are
    free. Cannot allocate. \n");
    }while(*frames_reqd>*noff);

    printf("\nEnter the reference string: ");scanf(" %[^\n]",
    reference_string);
}

//Split the string of references into integers
int refStringSplit(char* reference_string,int *
    page_references){
    int ctr=0;
    for(int i=0;reference_string[i];i++){
```

```
21          if(reference_string[i]>='0' && reference_string[i]<='
    9')
22              page_references[ctr++]=reference_string[i]-'0';
23      }
24      return ctr;
25  }
26
27  //Display the table of frames with each page reference
28  void displayTable(char page_ref_table[10][100],int
    frames_reqd,int col_ctr){
29
30      //Top line
31      for(int j=0;j<col_ctr;j++)
32          printf("______");
33      printf("\n");
34
35      for(int i=0;i<frames_reqd;i++){
36
37          for(int j=0;j<col_ctr;j++){
38              printf("|_%2c_|",page_ref_table[i][j]);
39          }
40          printf("\n");
41      }
42  }
43
44  //FIFO Algorithm
45  /*
46  page_fault: The page fault produced by the algorithm
    satisfying the given string.
47  page_references: Array of page references from the reference
    string.
48  page_ref_count: Number of references in the string.
49  page_ref_table: Table to keep track of all the changes in the
     frames.
50  row_ctr,col_ctr: Indices to traverse the page reference table
    .
51  insert_flag: Flag to indicate whether an updation has occured
     in the frames.
52
53  */
54  void FIFOAlgo(int frames_reqd,char *reference_string){
55      int page_fault=0;
56
57      //List of frames
58      Frame *LL=(Frame*)malloc(sizeof(Frame));
```

```
59      initialise(LL);
60
61      int page_references[100];
62      int page_ref_count=refStringSplit(reference_string,
        page_references);
63
64      char page_ref_table[10][100];
65      for(int i=0;i<10;i++)
66          for(int j=0;j<100;j++)
67              page_ref_table[i][j]=' ';
68
69      int row_ctr=0;
70      int col_ctr=0;
71
72      printf("\n FIFO Page Replacement Algorithm \n");
73      printf("\nThe reference string is: %-50s\n",
        reference_string);
74
75      printf("\n%-15s -> %-20s %5s -> %-12s\n","Page reference"
        ,"Memory"," ","Page fault");
76      int ctr=0;
77      int insert_flag=0;
78
79      /*
80      This loop fills the frame list for the first time, having
        number of nodes equal
81      to that of number of frames required by the process.
82      */
83      for(int i=0;i<frames_reqd && ctr<page_ref_count;i++){
84          //New Frame to be inserted
85          Frame* newFrame=(Frame*)malloc(sizeof(Frame));
86          initialise(newFrame);
87
88          insert_flag=0;
89
90          //If page reference already exists in the frames
91          if(!checkLL(LL,page_references[ctr])){
92              newFrame->page_no = page_references[ctr];
93              insert(LL,newFrame);
94              insert_flag=1;
95              page_fault++;
96          }
97          printf("\n%15d -> ",page_references[ctr]);
98          displayFrames(LL,frames_reqd);
99          printf("-> %-12d\n",page_fault);
```

```
100
101         //Updation of the page reference table
102         if(insert_flag){
103             page_ref_table[row_ctr][col_ctr]='0'+
    page_references[ctr];
104
105             for(int i=col_ctr+1;i<100;i++)
106                 page_ref_table[row_ctr][i] = page_ref_table[
    row_ctr][col_ctr];
107
108             if(row_ctr==frames_reqd-1)
109                 row_ctr=0;
110             else
111                 row_ctr++;
112             col_ctr++;
113         }
114
115         ctr++;
116     }
117
118     /*
119     This loop makes all the succesive updation to the frame
    list depending on the
120     specified algorithm.
121     */
122     for(;ctr<page_ref_count;ctr++){
123         Frame* newFrame=(Frame*)malloc(sizeof(Frame));
124         initialise(newFrame);
125
126         insert_flag=0;
127
128         if(!checkLL(LL,page_references[ctr])){
129             //Delete the first element and insert a new
    element, ie, FIFO
130             deleteFirst(LL);
131             newFrame->page_no = page_references[ctr];
132             insert(LL,newFrame);
133             insert_flag=1;
134             page_fault++;
135         }
136
137         printf("\n%15d -> ",page_references[ctr]);
138         //Display the updations to the frame list
139         displayFrames(LL,frames_reqd);
140         printf("-> %-12d\n",page_fault);
```

```
141
142         //Update the page reference table
143         if(insert_flag){
144             page_ref_table[row_ctr][col_ctr]='0'+
    page_references[ctr];
145
146             for(int i=col_ctr+1;i<100;i++)
147                 page_ref_table[row_ctr][i] = page_ref_table[
    row_ctr][col_ctr];
148
149             if(row_ctr==frames_reqd-1)
150                 row_ctr=0;
151             else
152                 row_ctr++;
153             col_ctr++;
154         }
155
156     }
157     displayTable(page_ref_table,frames_reqd,col_ctr);
158     printf("\nTotal number of Page faults:%d \n",page_fault);
159
160 }
161
162
163 //Optimal Position
164 /*
165 Logic:
166 1.Maintain an array where each element represents the index
      of the next occurence of the
167 corresponding element in the frame list.
168 2.Find the index of the maximum value in this array, which
      corresponds to the element that
169 is not used in the near future.
170 3.Pass the page_no of the element in that index in the frame
      list.
171 */
172
173 int OptimalPos(Frame *LL,int ctr,int page_references[100],int
     page_ref_count){
174     //Optimal position array
175     int opt_array[10];
176     for(int i=0;i<10;i++)
177         opt_array[i]=100;
178     int arr_ctr=0;
179     Frame *tmp=LL;
```

```
180     tmp=tmp->next;
181
182     //Compute occurences
183     while(tmp){
184         for(int i=ctr;i<page_ref_count;i++){
185             if(page_references[i]==tmp->page_no){
186                 opt_array[arr_ctr]=i;
187                 break;
188             }
189         }
190         tmp=tmp->next;
191         arr_ctr++;
192     }
193
194     //Compute index of maximum value
195     int maxid=0;
196     for(int i=0;i<arr_ctr;i++){
197         if(opt_array[i]>opt_array[maxid])
198             maxid=i;
199     }
200
201     tmp=LL;
202     tmp=tmp->next;
203     while(maxid){
204         tmp=tmp->next;maxid--;
205     }
206     return tmp->page_no;
207 }
208
209 //Optimal Algorithm
210 /*
211 page_fault: The page fault produced by the algorithm
        satisfying the given string.
212 page_references: Array of page references from the reference
        string.
213 page_ref_count: Number of references in the string.
214 page_ref_table: Table to keep track of all the changes in the
        frames.
215 row_ctr,col_ctr: Indices to traverse the page reference table
        .
216 insert_flag: Flag to indicate whether an updation has occured
        in the frames.
217
218 */
219 void OptimalAlgo(int frames_reqd,char *reference_string){
```

```
220     int page_fault=0;
221
222     Frame *LL=(Frame*)malloc(sizeof(Frame));
223     initialise(LL);
224
225     int page_references[100];
226     int page_ref_count=refStringSplit(reference_string,
    page_references);
227
228     char page_ref_table[10][100];
229     for(int i=0;i<10;i++)
230         for(int j=0;j<100;j++)
231             page_ref_table[i][j]=' ';
232
233     int row_ctr=0;
234     int col_ctr=0;
235
236     printf("\n Optimal Page Replacement Algorithm \n");
237     printf("\nThe reference string is: %-50s\n",
    reference_string);
238
239     printf("\n%-15s -> %-20s %5s -> %-12s\n","Page reference"
    ,"Memory"," ","Page fault");
240
241     int ctr=0;
242     int insert_flag=0;
243
244     /*
245     This loop fills the frame list for the first time, having
    number of nodes equal
246     to that of number of frames required by the process.
247     */
248     for(int i=0;i<frames_reqd && ctr<page_ref_count;i++){
249         //New frame to be inserted
250         Frame* newFrame=(Frame*)malloc(sizeof(Frame));
251         initialise(newFrame);
252
253         insert_flag=0;
254
255         //Check if element already exists in the frame list
256         if(!checkLL(LL,page_references[ctr])){
257             newFrame->page_no = page_references[ctr];
258             insert(LL,newFrame);
259             insert_flag=1;
260             page_fault++;
```

```
261         }
262         printf("\n%15d -> ",page_references[ctr]);
263         displayFrames(LL,frames_reqd);
264         printf("-> %-12d\n",page_fault);
265 
266         //Update the page reference table
267         if(insert_flag){
268             page_ref_table[row_ctr][col_ctr]='0'+
    page_references[ctr];
269 
270             for(int i=col_ctr+1;i<100;i++)
271                 page_ref_table[row_ctr][i] = page_ref_table[
    row_ctr][col_ctr];
272 
273                 row_ctr++;
274             col_ctr++;
275         }
276 
277         ctr++;
278     }
279 
280     /*
281     This loop makes all the succesive updation to the frame
    list depending on the
282     specified algorithm.
283     */
284     for(;ctr<page_ref_count;ctr++){
285 
286         insert_flag=0;
287 
288         int page_no;
289         if(!checkLL(LL,page_references[ctr])){
290             page_no=OptimalPos(LL,ctr,page_references,
    page_ref_count);
291             ReplacePos(LL,page_no, page_references[ctr]);
292             insert_flag=1;
293             page_fault++;
294         }
295         printf("\n%15d -> ",page_references[ctr]);
296         displayFrames(LL,frames_reqd);
297         printf("-> %-12d\n",page_fault);
298 
299         //Update page reference table
300         if(insert_flag){
301             for(int i=0;i<frames_reqd;i++){
```

```
302                 if(page_ref_table[i][col_ctr]=='0'+page_no){
303                     row_ctr=i;break;
304 
305                 }
306             }
307 
308             page_ref_table[row_ctr][col_ctr]='0'+
    page_references[ctr];
309 
310             for(int i=col_ctr+1;i<100;i++)
311                 page_ref_table[row_ctr][i] = page_ref_table[
    row_ctr][col_ctr];
312             col_ctr++;
313         }
314 
315     }
316     displayTable(page_ref_table,frames_reqd,col_ctr);
317     printf("\nTotal number of Page faults:%d \n",page_fault);
318 
319 }
320 
321 
322 //Least Recently Used Position
323 /*
324 Logic:
325 1.Maintain an array where each element represents the index
    of the previous occurence of the
326 corresponding element in the frame list.
327 2.Find the index of the minimum value in this array, which
    corresponds to the element that
328 is not used in the near future.
329 3.Pass the page_no of the element in that index in the frame
    list.
330 */
331 
332 int LRUPos(Frame *LL,int ctr,int page_references[100],int
    page_ref_count){
333     //LRU Position array
334     int lru_array[10];
335     for(int i=0;i<10;i++)
336         lru_array[i]=100;
337     int arr_ctr=0;
338     Frame *tmp=LL;
339     tmp=tmp->next;
340 
```

```
341     //Compute LRU positions
342     while(tmp){
343         for(int i=0;i<ctr;i++){
344             if(page_references[i]==tmp->page_no){
345                 lru_array[arr_ctr]=i;
346             }
347         }
348         tmp=tmp->next;
349         arr_ctr++;
350     }
351 
352     //Compute index of minimum value
353     int minid=0;
354     for(int i=0;i<arr_ctr;i++){
355         if(lru_array[i]<lru_array[minid])
356             minid=i;
357     }
358 
359     tmp=LL;
360     tmp=tmp->next;
361     while(minid){
362         tmp=tmp->next;minid--;
363     }
364     return tmp->page_no;
365 }
366 
367 //Least Recently Used Algorithm
368 /*
369 page_fault: The page fault produced by the algorithm
        satisfying the given string.
370 page_references: Array of page references from the reference
        string.
371 page_ref_count: Number of references in the string.
372 page_ref_table: Table to keep track of all the changes in the
         frames.
373 row_ctr,col_ctr: Indices to traverse the page reference table
        .
374 insert_flag: Flag to indicate whether an updation has occured
         in the frames.
375 
376 */
377 
378 void LRUAlgo(int frames_reqd,char *reference_string){
379 
380     int page_fault=0;
```

```
381
382     Frame *LL=(Frame*)malloc(sizeof(Frame));
383     initialise(LL);
384
385     int page_references[100];
386     int page_ref_count=refStringSplit(reference_string,
    page_references);
387
388     char page_ref_table[10][100];
389     for(int i=0;i<10;i++)
390         for(int j=0;j<100;j++)
391             page_ref_table[i][j]=' ';
392
393     int row_ctr=0;
394     int col_ctr=0;
395
396     printf("\n LRU Page Replacement Algorithm \n");
397     printf("\nThe reference string is: %-50s\n",
    reference_string);
398
399     printf("\n%-15s -> %-20s %5s -> %-12s\n","Page reference"
    ,"Memory"," ","Page fault");
400
401     int ctr=0;
402     int insert_flag=0;
403
404     /*
405     This loop fills the frame list for the first time, having
     number of nodes equal
406     to that of number of frames required by the process.
407     */
408     for(int i=0;i<frames_reqd && ctr<page_ref_count;i++){
409         //New Frame to be inserted
410         Frame* newFrame=(Frame*)malloc(sizeof(Frame));
411         initialise(newFrame);
412
413         insert_flag=0;
414
415         //Check if element is already present in frame list
416         if(!checkLL(LL,page_references[ctr])){
417             newFrame->page_no = page_references[ctr];
418             insert(LL,newFrame);
419             insert_flag=1;
420             page_fault++;
421         }
```

```
422         printf("\n%15d -> ",page_references[ctr]);
423         displayFrames(LL,frames_reqd);
424         printf("-> %-12d\n",page_fault);
425 
426         //Update the page reference table
427         if(insert_flag){
428             page_ref_table[row_ctr][col_ctr]='0'+
    page_references[ctr];
429 
430             for(int i=col_ctr+1;i<100;i++)
431                 page_ref_table[row_ctr][i] = page_ref_table[
    row_ctr][col_ctr];
432 
433                 row_ctr++;
434             col_ctr++;
435         }
436 
437         ctr++;
438      }
439 
440      /*
441      This loop makes all the succesive updation to the frame
    list depending on the
442      specified algorithm.
443      */
444      for(;ctr<page_ref_count;ctr++){
445 
446         insert_flag=0;
447 
448         int page_no;
449         if(!checkLL(LL,page_references[ctr])){
450             page_no=LRUPos(LL,ctr,page_references,
    page_ref_count);
451             ReplacePos(LL,page_no, page_references[ctr]);
452             insert_flag=1;
453             page_fault++;
454         }
455         printf("\n%15d -> ",page_references[ctr]);
456         displayFrames(LL,frames_reqd);
457         printf("-> %-12d\n",page_fault);
458 
459         //Update the page reference table
460         if(insert_flag){
461             for(int i=0;i<frames_reqd;i++){
462                 if(page_ref_table[i][col_ctr]=='0'+page_no){
```

```
463                     row_ctr=i;break;
464
465                 }
466             }
467
468             page_ref_table[row_ctr][col_ctr]='0'+
    page_references[ctr];
469
470             for(int i=col_ctr+1;i<100;i++)
471                 page_ref_table[row_ctr][i] = page_ref_table[
    row_ctr][col_ctr];
472             col_ctr++;
473         }
474
475     }
476     displayTable(page_ref_table,frames_reqd,col_ctr);
477     printf("\nTotal number of Page faults:%d \n",page_fault);
478 }
479
480 //Least Frequently Used Position
481 /*
482 Logic:
483 1.Maintain an array where each element represents the
    frequency of the occurences of the
484 corresponding element in the frame list.
485 2.Find the index of the minimum value in this array, which
    corresponds to the element that
486 is not used in the near future.
487 3.Pass the page_no of the element in that index in the frame
    list.
488 */
489
490 int LFUPos(Frame *LL,int ctr,int page_references[100],int
    page_ref_count){
491     //LFU array
492     int lfu_array[10];
493     for(int i=0;i<10;i++)
494         lfu_array[i]=0;
495     int arr_ctr=0;
496     Frame *tmp=LL;
497     tmp=tmp->next;
498
499     //Compute frequencies
500     while(tmp){
501         for(int i=ctr;i<page_ref_count;i++){
```

```
502             if(page_references[i]==tmp->page_no){
503                 lfu_array[arr_ctr]++;
504             }
505         }
506         tmp=tmp->next;
507         arr_ctr++;
508     }
509 
510     //Compute index of minimum element
511     int minid=0;
512     for(int i=0;i<arr_ctr;i++){
513         if(lfu_array[i]<lfu_array[minid])
514             minid=i;
515     }
516 
517     tmp=LL;
518     tmp=tmp->next;
519     while(minid){
520         tmp=tmp->next;minid--;
521     }
522     return tmp->page_no;
523 }
524 
525 //Least Frequently Used Algorithm
526 /*
527 page_fault: The page fault produced by the algorithm
        satisfying the given string.
528 page_references: Array of page references from the reference
        string.
529 page_ref_count: Number of references in the string.
530 page_ref_table: Table to keep track of all the changes in the
         frames.
531 row_ctr,col_ctr: Indices to traverse the page reference table
        .
532 insert_flag: Flag to indicate whether an updation has occured
         in the frames.
533 
534 */
535 
536 void LFUAlgo(int frames_reqd,char *reference_string){
537 
538     int page_fault=0;
539 
540     Frame *LL=(Frame*)malloc(sizeof(Frame));
541     initialise(LL);
```

```
542
543     int page_references[100];
544     int page_ref_count=refStringSplit(reference_string,
        page_references);
545
546     char page_ref_table[10][100];
547     for(int i=0;i<10;i++)
548         for(int j=0;j<100;j++)
549             page_ref_table[i][j]=' ';
550
551     int row_ctr=0;
552     int col_ctr=0;
553
554     printf("\n LFU Page Replacement Algorithm \n");
555     printf("\nThe reference string is: %-50s\n",
        reference_string);
556
557     printf("\n%-15s -> %-20s %5s -> %-12s\n","Page reference"
        ,"Memory"," ","Page fault");
558
559     int ctr=0;
560     int insert_flag=0;
561
562     /*
563     This loop fills the frame list for the first time, having
        number of nodes equal
564     to that of number of frames required by the process.
565     */
566     for(int i=0;i<frames_reqd && ctr<page_ref_count;i++){
567         //New Frame to be inserted
568         Frame* newFrame=(Frame*)malloc(sizeof(Frame));
569         initialise(newFrame);
570
571         insert_flag=0;
572
573         //Check if element is already in frame list
574         if(!checkLL(LL,page_references[ctr])){
575             newFrame->page_no = page_references[ctr];
576             insert(LL,newFrame);
577             insert_flag=1;
578             page_fault++;
579         }
580         printf("\n%15d -> ",page_references[ctr]);
581         displayFrames(LL,frames_reqd);
582         printf("-> %-12d\n",page_fault);
```

```
583
584         //Update the page reference table
585         if(insert_flag){
586             page_ref_table[row_ctr][col_ctr]='0'+
    page_references[ctr];
587
588             for(int i=col_ctr+1;i<100;i++)
589                 page_ref_table[row_ctr][i] = page_ref_table[
    row_ctr][col_ctr];
590
591                 row_ctr++;
592             col_ctr++;
593         }
594
595         ctr++;
596      }
597
598      /*
599      This loop makes all the succesive updation to the frame
    list depending on the
600      specified algorithm.
601      */
602      for(;ctr<page_ref_count;ctr++){
603
604         insert_flag=0;
605
606         int page_no;
607         if(!checkLL(LL,page_references[ctr])){
608             page_no=LFUPos(LL,ctr,page_references,
    page_ref_count);
609             ReplacePos(LL,page_no, page_references[ctr]);
610             insert_flag=1;
611             page_fault++;
612         }
613         printf("\n%15d -> ",page_references[ctr]);
614         displayFrames(LL,frames_reqd);
615         printf("-> %-12d\n",page_fault);
616
617         //Update the page reference table
618         if(insert_flag){
619             for(int i=0;i<frames_reqd;i++){
620                 if(page_ref_table[i][col_ctr]=='0'+page_no){
621                     row_ctr=i;break;
622
623                 }
```

```
624             }
625 
626             page_ref_table[row_ctr][col_ctr]='0'+
    page_references[ctr];
627 
628             for(int i=col_ctr+1;i<100;i++)
629                 page_ref_table[row_ctr][i] = page_ref_table[
    row_ctr][col_ctr];
630             col_ctr++;
631         }
632 
633     }
634     displayTable(page_ref_table,frames_reqd,col_ctr);
635     printf("\nTotal number of Page faults:%d \n",page_fault);
636 }
637 
638 void main(){
639     //Number of free frames
640     int noff;
641     //Frames required by process
642     int frames_reqd;
643     //Page reference string
644     char *reference_string=(char*)malloc(sizeof(100));
645 
646     int option;
647     do{
648         printf("\n Choose operation: \n 1.Read data \n 2.FIFO
     algorithm ");
649         printf("\n 3.Optimal algorithm \n 4.LRU algorithm \n
    5.LFU algorithm ");
650         printf("\n 0.Exit \n Your choice: ");scanf("%d",&
    option);
651 
652         if(option==1){
653             acceptData(&noff,&frames_reqd,reference_string);
654         }
655         else if(option==2){
656             FIFOAlgo(frames_reqd,reference_string);
657         }
658         else if(option==3){
659             OptimalAlgo(frames_reqd,reference_string);
660         }
661         else if(option==4){
662             LRUAlgo(frames_reqd,reference_string);
663         }
```

```
664         else if(option==5){
665             LFUAlgo(frames_reqd,reference_string);
666         }
667         else if(option){
668             printf("\n Invalid option. \n");
669         }
670         else;
671     }while(option);
672 
673 }
```

Output:

```
1 
2  Choose operation:
3  1.Read data
4  2.FIFO algorithm
5  3.Optimal algorithm
6  4.LRU algorithm
7  5.LFU algorithm
8  0.Exit
9  Your choice: 1
10 
11 Enter the number of free frames: 10
12 
13 Enter the number of frames required: 3
14 
15 Enter the reference string: 7 0 1 2 0 3 0 4 2 3 0 3
16 
17  Choose operation:
18  1.Read data
19  2.FIFO algorithm
20  3.Optimal algorithm
21  4.LRU algorithm
22  5.LFU algorithm
23  0.Exit
24  Your choice: 2
25 
26  FIFO Page Replacement Algorithm
27 
28 The reference string is: 7 0 1 2 0 3 0 4 2 3 0 3
29 
30 Page reference  -> Memory                    -> Page fault
31 
32               7 -> 7    -    -               -> 1
```

```
                0 -> 7    0    -                   -> 2

                1 -> 7    0    1                   -> 3

                2 -> 0    1    2                   -> 4

                0 -> 0    1    2                   -> 4

                3 -> 1    2    3                   -> 5

                0 -> 2    3    0                   -> 6

                4 -> 3    0    4                   -> 7

                2 -> 0    4    2                   -> 8

                3 -> 4    2    3                   -> 9

                0 -> 2    3    0                   -> 10

                3 -> 2    3    0                   -> 10
-------------------------------------------------------------
|_ 7_||_ 7_||_ 7_||_ 2_||_ 2_||_ 2_||_ 4_||_ 4_||_ 4_||_ 0_|
|_  _||_ 0_||_ 0_||_ 0_||_ 3_||_ 3_||_ 3_||_ 2_||_ 2_||_ 2_|
|_  _||_  _||_ 1_||_ 1_||_ 1_||_ 0_||_ 0_||_ 0_||_ 3_||_ 3_|

Total number of Page faults:10

 Choose operation:
 1.Read data
 2.FIFO algorithm
 3.Optimal algorithm
 4.LRU algorithm
 5.LFU algorithm
 0.Exit
 Your choice: 3

 Optimal Page Replacement Algorithm

The reference string is: 7 0 1 2 0 3 0 4 2 3 0 3

Page reference  -> Memory                       -> Page fault

                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 -> 0    4    3                -> 7

                3 -> 0    4    3                -> 7
-----------------------------------------
|_ 7_||_ 7_||_ 7_||_ 2_||_ 2_||_ 2_||_ 0_|
|_  _||_ 0_||_ 0_||_ 0_||_ 0_||_ 4_||_ 4_|
|_  _||_  _||_ 1_||_ 1_||_ 3_||_ 3_||_ 3_|

Total number of Page faults:7

 Choose operation:
 1.Read data
 2.FIFO algorithm
 3.Optimal algorithm
 4.LRU algorithm
 5.LFU algorithm
 0.Exit
 Your choice: 4

 LRU Page Replacement Algorithm

The reference string is: 7 0 1 2 0 3 0 4 2 3 0 3

Page reference  -> Memory                    -> Page fault

                7 -> 7    -    -                -> 1
```

```
123
124                0 -> 7    0    -                   -> 2
125
126                1 -> 7    0    1                   -> 3
127
128                2 -> 2    0    1                   -> 4
129
130                0 -> 2    0    1                   -> 4
131
132                3 -> 2    0    3                   -> 5
133
134                0 -> 2    0    3                   -> 5
135
136                4 -> 4    0    3                   -> 6
137
138                2 -> 4    0    2                   -> 7
139
140                3 -> 4    3    2                   -> 8
141
142                0 -> 0    3    2                   -> 9
143
144                3 -> 0    3    2                   -> 9
145 ------------------------------------------------------
146 |_ 7_||_ 7_||_ 7_||_ 2_||_ 2_||_ 4_||_ 4_||_ 4_||_ 0_|
147 |_  _||_ 0_||_ 0_||_ 0_||_ 0_||_ 0_||_ 0_||_ 3_||_ 3_|
148 |_  _||_  _||_ 1_||_ 1_||_ 3_||_ 3_||_ 2_||_ 2_||_ 2_|
149
150 Total number of Page faults:9
151
152  Choose operation:
153  1.Read data
154  2.FIFO algorithm
155  3.Optimal algorithm
156  4.LRU algorithm
157  5.LFU algorithm
158  0.Exit
159  Your choice: 5
160
161  LFU Page Replacement Algorithm
162
163 The reference string is: 7 0 1 2 0 3 0 4 2 3 0 3
164
165 Page reference  -> Memory                      -> Page fault
166
167                7 -> 7    -    -                   -> 1
```

```
168
169                 0 -> 7    0    -                   -> 2
170
171                 1 -> 7    0    1                   -> 3
172
173                 2 -> 2    0    1                   -> 4
174
175                 0 -> 2    0    1                   -> 4
176
177                 3 -> 2    0    3                   -> 5
178
179                 0 -> 2    0    3                   -> 5
180
181                 4 -> 4    0    3                   -> 6
182
183                 2 -> 2    0    3                   -> 7
184
185                 3 -> 2    0    3                   -> 7
186
187                 0 -> 2    0    3                   -> 7
188
189                 3 -> 2    0    3                   -> 7
190 ----------------------------------------
191 |_ 7_||_ 7_||_ 7_||_ 2_||_ 2_||_ 4_||_ 2_|
192 |_  _||_ 0_||_ 0_||_ 0_||_ 0_||_ 0_||_ 0_|
193 |_  _||_  _||_ 1_||_ 1_||_ 3_||_ 3_||_ 3_|
194
195 Total number of Page faults:7
196
197  Choose operation:
198  1.Read data
199  2.FIFO algorithm
200  3.Optimal algorithm
201  4.LRU algorithm
202  5.LFU algorithm
203  0.Exit
204  Your choice: 0
```