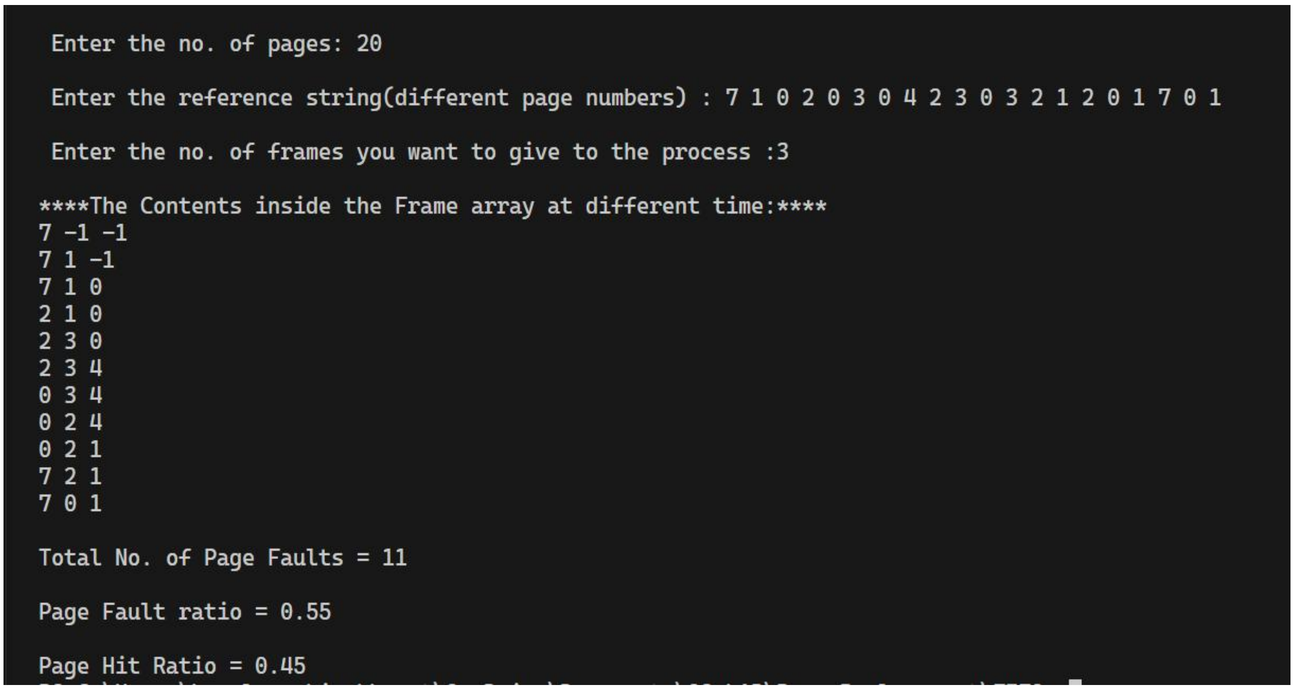
Name : Kunal Sachin Kharat
Roll no : 33
Class : CSAIML-A

Q.) Implement Page Replacement Algorithms:

1. FIFO(First In First Out):

```c
//C Program to Implement the FIFO(First In First Out) Page replacement Algorithm
//Time Complexity = O(n)
//Space Complexity= O(no of frames + size of Page Table)

#include<stdio.h>
#include<stdbool.h>
#include<string.h>
struct PageTable
{
    int frame_no;
    bool valid;
};
//Function to check if referenced/asked page is already present in frame[] or not
//Returns true if page is already present else returns false
bool isPagePresent(struct PageTable PT[],int page,int n)
{
    if(PT[page].valid == 1)
       return true;
    return false;
}
//Function to update the page table
//Return Nothing
void updatePageTable(struct PageTable PT[],int page,int fr_no,int status)
{
    PT[page].valid=status;
    //if(status == 1 )
       PT[page].frame_no=fr_no;
}
//Function to print the frame contents
//Return nothing
void printFrameContents(int frame[],int no_of_frames)
{
    for(int i=0;i<no_of_frames;i++)
      printf("%d ",frame[i]);
    printf("\n");
}
int main()
{
    int i,n,no_of_frames,page_fault=0,current=0;
    bool flag=false;
    printf("\n Enter the no. of pages: ");
    scanf("%d",&n);
    //create reference string array
    int reference_string[n];
    printf("\n Enter the reference string(different page numbers) : ");
    for(int i=0;i<n;i++)
     scanf("%d",&reference_string[i]);
    printf("\n Enter the no. of frames you want to give to the process :");
    scanf("%d",&no_of_frames);
    //create frame array to store the pages at different point of times
    int frame[no_of_frames];
    memset(frame,-1,no_of_frames*sizeof(int));
    struct PageTable PT[50] ; //asume page table can have entries for page 0 to 49
    for(int i=0;i<50;i++)
      PT[i].valid=0;

    printf("\n****The Contents inside the Frame array at different time:****\n");
    for(int i=0;i<n;i++)
    {
      //search the ith page in all allocated frames
      if( ! (isPagePresent(PT,reference_string[i],n)))
      {
         page_fault++;          // Increase the count of page fault
         if(flag==false && current < no_of_frames)
         {
                frame[current]=reference_string[i];
                printFrameContents(frame,no_of_frames);
                updatePageTable(PT,reference_string[i],current,1);
                current = current + 1;
                if(current == no_of_frames)
                {
                   current=0;
                   flag=true;  // so that we do not come to this if block again
                }

         }

         else //frame are full , APPLY FIFO
         {
            //find the FIFO page (victim page) to replace;
            //The page pointed by current_head is FIFO page (victim page), so need to
find it :)
            //mark that page as INVALID as in Page Table
            //set invalid frame no as -1 or anything ( as function needs this parameter),
                updatePageTable(PT,frame[current], -1 ,0);
                frame[current]=reference_string[i];
                printFrameContents(frame,no_of_frames);
                updatePageTable(PT,reference_string[i],current,1);
                current = ( current + 1)% no_of_frames;
         }
      } //end of outer if
    }   //end of for loop


   printf("\nTotal No. of Page Faults = %d\n",page_fault);
   printf("\nPage Fault ratio = %.2f\n",(float)page_fault/n);
   printf("\nPage Hit Ratio = %.2f\n",(float)(n- page_fault)/n);
   return 0;
}
```

Output:

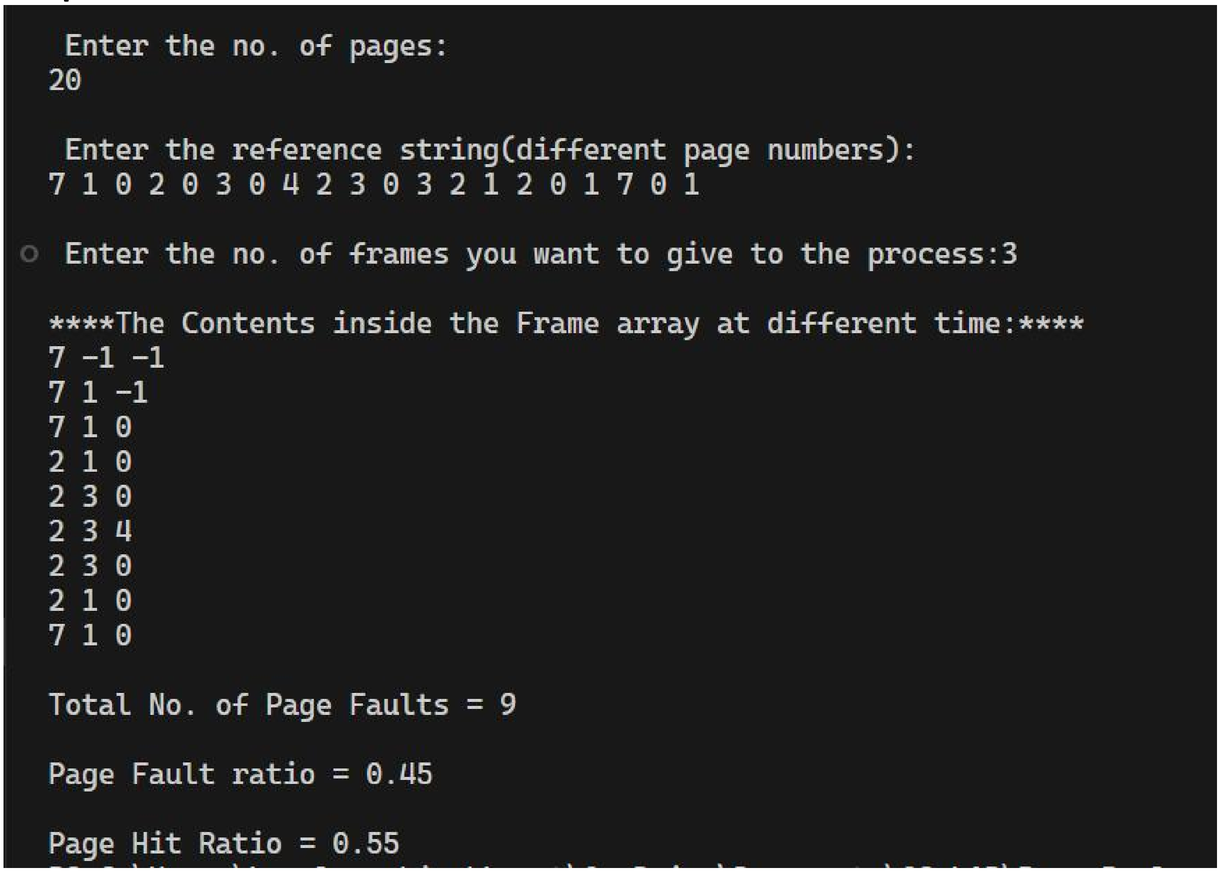


```
 Enter the no. of pages: 20

 Enter the reference string(different page numbers) : 7 1 0 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

 Enter the no. of frames you want to give to the process :3

****The Contents inside the Frame array at different time:****
7 -1 -1
7 1 -1
7 1 0
2 1 0
2 3 0
2 3 4
0 3 4
0 2 4
0 2 1
7 2 1
7 0 1

Total No. of Page Faults = 11

Page Fault ratio = 0.55

Page Hit Ratio = 0.45
```

2. Optimal Page Replacement:

```c
#include<stdio.h>
#include<stdbool.h>
#include<string.h>
#include<limits.h>

struct PageTable
{
    int frame_no;
    int last_time_of_access;
    bool valid;
};
bool isPagePresent(struct PageTable PT[], int page)
{
    if(PT[page].valid == 1)
        return true;
    return false;
}
void updatePageTable(struct PageTable PT[], int page, int fr_no, int status, int
access_time)
{
    PT[page].valid=status;
    if(status == 1 )
    {
      PT[page].last_time_of_access =  access_time;
      PT[page].frame_no=fr_no;
    }
}
void printFrameContents(int frame[], int no_of_frames)
{
    for(int i = 0; i < no_of_frames; i++)
        printf("%d ", frame[i]);
    printf("\n");
}
int searchOptimalPage(struct PageTable PT[], int reference_string[], int n, int frame[],
int no_of_frames, int start)
{
    int farthest = start;
    int victim_page_index = -1;
    for(int i = 0; i < no_of_frames; i++)
    {
        int j;
        for(j = start + 1; j < n; j++)
        {
            if(frame[i] == reference_string[j])
            {
                if(j > farthest)
                {
                    farthest = j;
                    victim_page_index = i;
                }
                break;
            }
        }
        if(j == n)
            return i;
    }
    if(victim_page_index == -1)
        return 0;
    return victim_page_index;
}
int main()
{
    int i, n, no_of_frames, page_fault = 0, current = 0;
    bool flag = false;
    printf("\n Enter the no. of pages:\n");
    scanf("%d", &n);
    int reference_string[n];
    printf("\n Enter the reference string(different page numbers):\n");
    for(int i = 0; i < n; i++)
        scanf("%d", &reference_string[i]);
    printf("\n Enter the no. of frames you want to give to the process:");
    scanf("%d", &no_of_frames);
    int frame[no_of_frames];
    memset(frame, -1, no_of_frames * sizeof(int));
    struct PageTable PT[50];
    for(int i = 0; i < 50; i++)
        PT[i].valid = 0;

    printf("\n****The Contents inside the Frame array at different time:****\n");
    for(int i = 0; i < n; i++)
    {
        if(!(isPagePresent(PT, reference_string[i])))
        {
            page_fault++;
            if(flag == false && current < no_of_frames)
            {
                frame[current] = reference_string[i];
                printFrameContents(frame, no_of_frames);
                updatePageTable(PT, reference_string[i], current, 1, i);
                current = current + 1;
                if(current == no_of_frames)
                    flag = true;
            }
            else
            {
                int victim_page_index = searchOptimalPage(PT, reference_string, n, frame,
no_of_frames, i);
                updatePageTable(PT, frame[victim_page_index], -1, 0, i);
                frame[victim_page_index] = reference_string[i];
                printFrameContents(frame, no_of_frames);
                updatePageTable(PT, reference_string[i], victim_page_index, 1, i);
            }
        }
        PT[reference_string[i]].last_time_of_access =  i;
    }
    printf("\nTotal No. of Page Faults = %d\n", page_fault);
    printf("\nPage Fault ratio = %.2f\n", (float)page_fault/n);
    printf("\nPage Hit Ratio = %.2f\n", (float)(n - page_fault)/n);
    return 0;
}
```

Output:



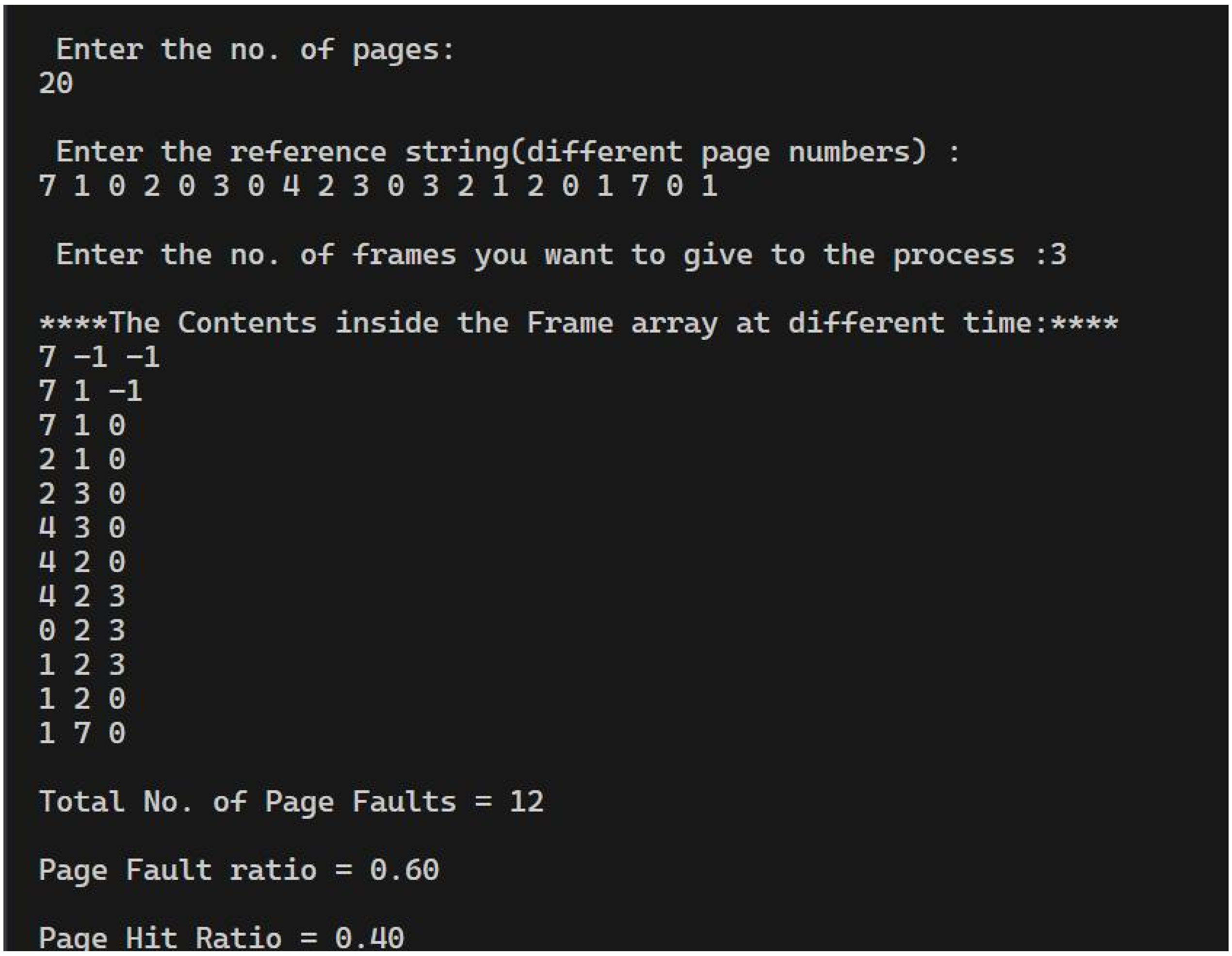
```
 Enter the no. of pages:
20

 Enter the reference string(different page numbers):
7 1 0 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

 Enter the no. of frames you want to give to the process:3

****The Contents inside the Frame array at different time:****
7 -1 -1
7 1 -1
7 1 0
2 1 0
2 3 0
2 3 4
2 3 0
2 1 0
7 1 0

Total No. of Page Faults = 9

Page Fault ratio = 0.45

Page Hit Ratio = 0.55
```

3. Least Recently Used:

```
//C Program to Implement the LRU(Least Recently Used) Page replacement Algorithm

#include<stdio.h>
#include<stdbool.h>
#include<string.h>
#include<limits.h>
struct PageTable
{
    int frame_no;
    int last_time_of_access;
    bool valid;
};
//Function to check if referenced/asked page is already present in frame[] or not
//Returns true if page is already present else returns false
bool isPagePresent(struct PageTable PT[],int page)
{
    if(PT[page].valid == 1)
       return true;
    return false;
}
//Function to update the page table
//Return Nothing
void updatePageTable(struct PageTable PT[],int page,int fr_no,int status,int access_time)
{
    PT[page].valid=status;
    if(status == 1 )
    {
      PT[page].last_time_of_access =  access_time;
      PT[page].frame_no=fr_no;
    }
}
//Function to print the frame contents
//Return nothing
void printFrameContents(int frame[],int no_of_frames)
{
    for(int i=0;i<no_of_frames;i++)
      printf("%d ",frame[i]);
    printf("\n");
}
//Function to find the victim page index in frame[]
//Return that LRU page index using call by address
void searchLRUPage(struct PageTable PT[], int frame[], int no_of_frames, int
*LRU_page_index)
{
    int min = INT_MAX;
    for(int i=0; i<no_of_frames;i++)
    {
       if(PT[frame[i]].last_time_of_access < min)
       {
           min = PT[frame[i]].last_time_of_access;
           *LRU_page_index = i;
       }
    }
}
int main()
{
    int i,n,no_of_frames,page_fault=0,current=0;
    bool flag=false;
    printf("\n Enter the no. of pages:\n");
    scanf("%d",&n);
    //create reference string array
    int reference_string[n];
    printf("\n Enter the reference string(different page numbers) :\n");
    for(int i=0;i<n;i++)
     scanf("%d",&reference_string[i]);
    printf("\n Enter the no. of frames you want to give to the process :");
    scanf("%d",&no_of_frames);
    //create frame array to store the pages at different point of times
    int frame[no_of_frames];
    memset(frame,-1,no_of_frames*sizeof(int));
    struct PageTable PT[50] ; //asume page table can have entries for page 0 to 49
    for(int i=0;i<50;i++)
      PT[i].valid=0;

    printf("\n****The Contents inside the Frame array at different time:****\n");
    for(int i=0;i<n;i++)
    {
      //search the ith page in all allocated frames
      if( ! (isPagePresent(PT,reference_string[i])))
      {
         page_fault++;      // Increase the count of page fault
         if(flag==false && current < no_of_frames)
         {

            frame[current]=reference_string[i];
            printFrameContents(frame,no_of_frames);
            updatePageTable(PT,reference_string[i],current,1,i);

            current = current + 1;
            if(current == no_of_frames)
            {
                //current=0;
                flag=true;
            }

         }

         else //frame are full , APPLY LRU Algo
         {
            //search the LRU page( victim page) with the help of PT
            //mark that page as INVALID in Page Table
            int LRU_page_index;
            searchLRUPage(PT,frame,no_of_frames,&LRU_page_index);
            updatePageTable(PT,frame[LRU_page_index], -1 ,0,i);  //send invalid frame_no
=-1
            frame[LRU_page_index]=reference_string[i];
            printFrameContents(frame,no_of_frames);
            //Update PT
            updatePageTable(PT,reference_string[i],LRU_page_index,1,i);
         }
      }
      //Update the Page Access time for reference_string[i]
      PT[reference_string[i]].last_time_of_access =  i;
    }  //end of for loop


   printf("\nTotal No. of Page Faults = %d\n",page_fault);
```

```
    printf("\nPage Fault ratio = %.2f\n",(float)page_fault/n);
    printf("\nPage Hit Ratio = %.2f\n",(float)(n- page_fault)/n);
    return 0;
}
```

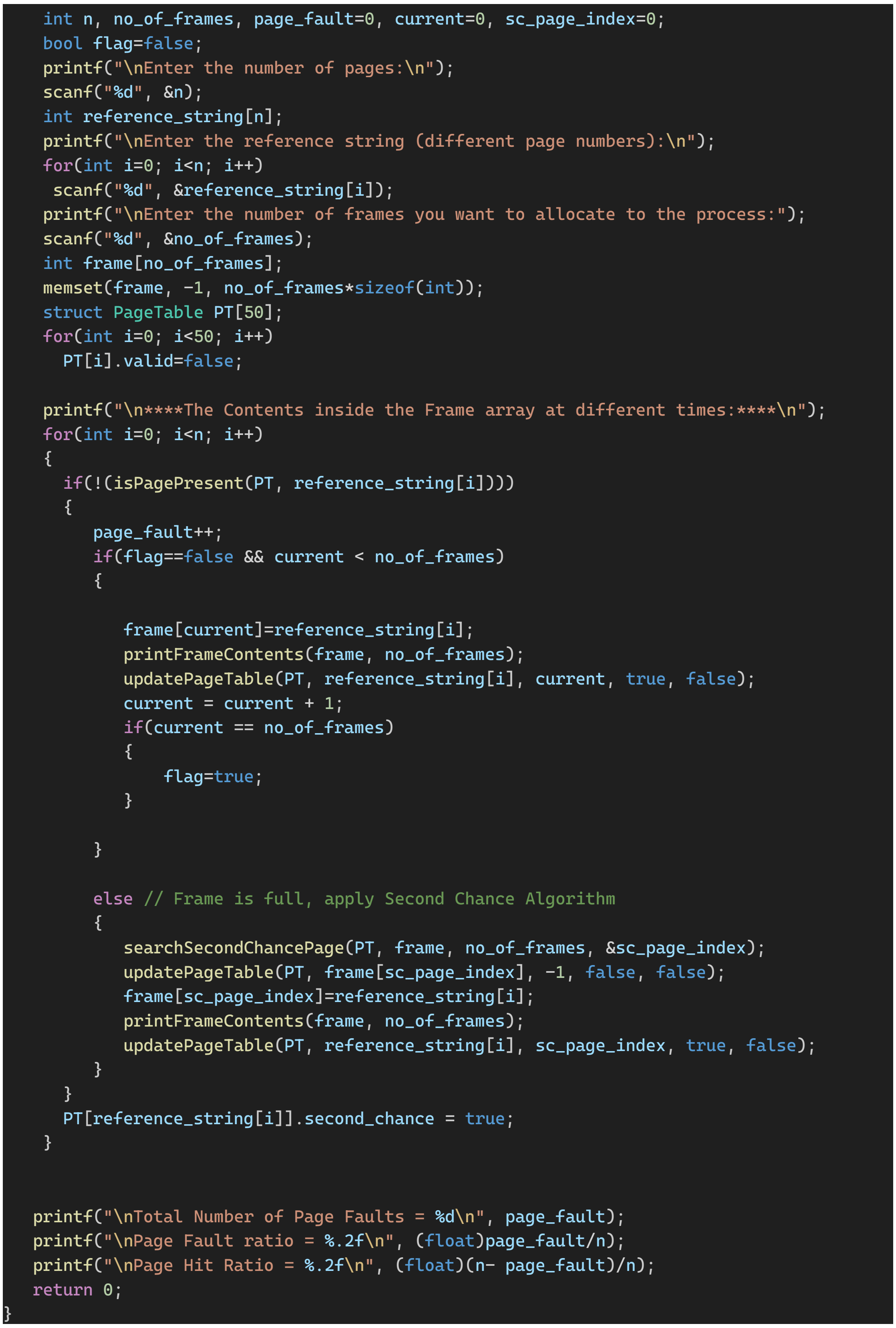
Output:



```
 Enter the no. of pages:
20

 Enter the reference string(different page numbers) :
7 1 0 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

 Enter the no. of frames you want to give to the process :3

****The Contents inside the Frame array at different time:****
7 -1 -1
7 1 -1
7 1 0
2 1 0
2 3 0
4 3 0
4 2 0
4 2 3
0 2 3
1 2 3
1 2 0
1 7 0

Total No. of Page Faults = 12

Page Fault ratio = 0.60

Page Hit Ratio = 0.40
```

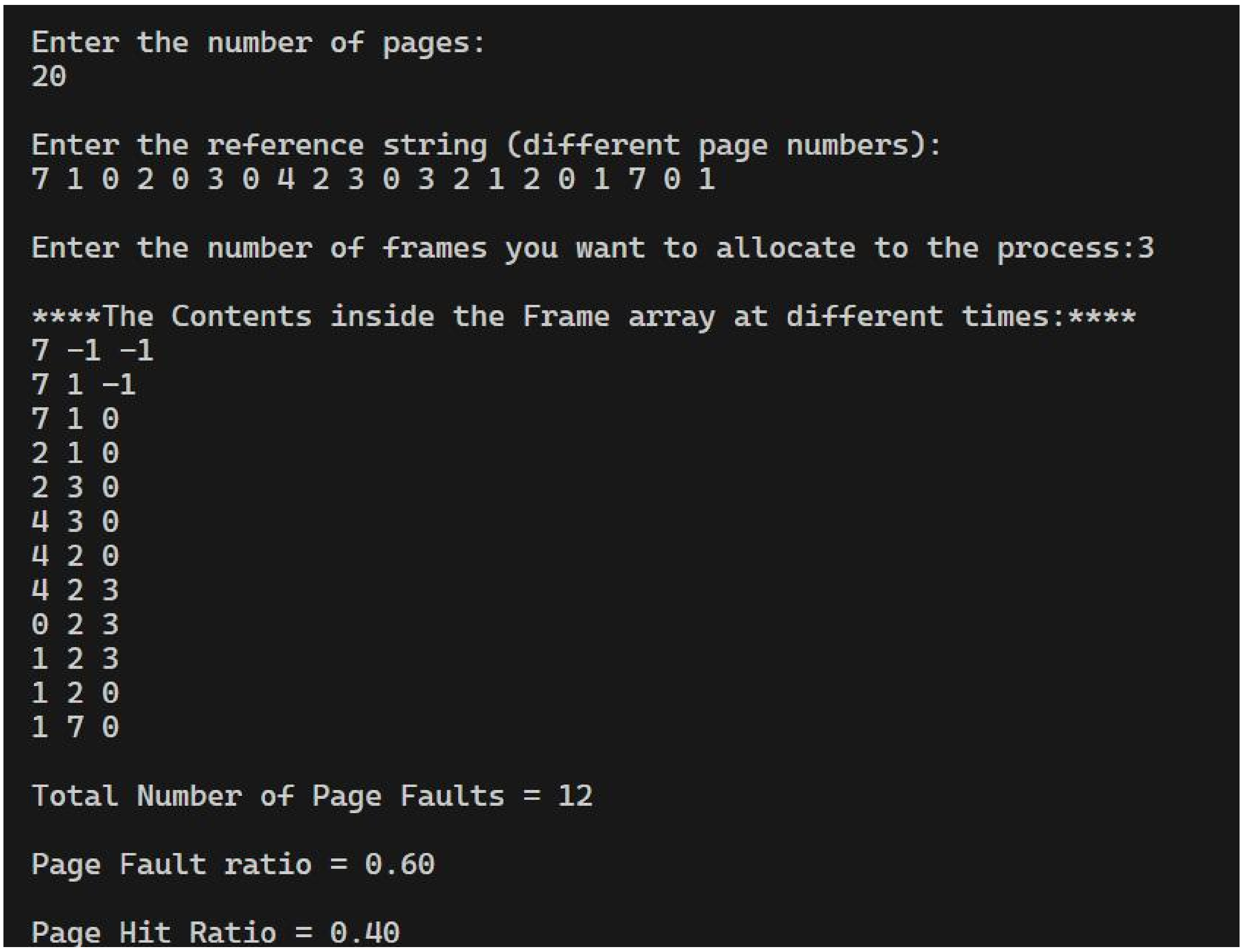
4. Second Chance (Clock)

```c
#include<stdio.h>
#include<stdbool.h>
#include<string.h>
#include<limits.h>

struct PageTable
{
    int frame_no;
    bool valid;
    bool second_chance;
};
bool isPagePresent(struct PageTable PT[],int page)
{
    if(PT[page].valid == true)
       return true;
    return false;
}
void updatePageTable(struct PageTable PT[],int page,int fr_no,bool status,bool sc)
{
    PT[page].valid=status;
    PT[page].second_chance=sc;
    PT[page].frame_no=fr_no;
}
void printFrameContents(int frame[],int no_of_frames)
{
    for(int i=0;i<no_of_frames;i++)
      printf("%d ",frame[i]);
    printf("\n");
}
void searchSecondChancePage(struct PageTable PT[], int frame[], int no_of_frames, int
*sc_page_index)
{
    while (true) {
        if (!PT[frame[*sc_page_index]].second_chance) {
            // Found a page with second chance bit unset
            return;
        }
        // Set second chance bit to false and move to the next page in the clock
        PT[frame[*sc_page_index]].second_chance = false;
        *sc_page_index = (*sc_page_index + 1) % no_of_frames;
    }
}
int main()
{
```



```c
    int n, no_of_frames, page_fault=0, current=0, sc_page_index=0;
    bool flag=false;
    printf("\nEnter the number of pages:\n");
    scanf("%d", &n);
    int reference_string[n];
    printf("\nEnter the reference string (different page numbers):\n");
    for(int i=0; i<n; i++)
     scanf("%d", &reference_string[i]);
    printf("\nEnter the number of frames you want to allocate to the process:");
    scanf("%d", &no_of_frames);
    int frame[no_of_frames];
    memset(frame, -1, no_of_frames*sizeof(int));
    struct PageTable PT[50];
    for(int i=0; i<50; i++)
      PT[i].valid=false;

    printf("\n****The Contents inside the Frame array at different times:****\n");
    for(int i=0; i<n; i++)
    {
      if(!(isPagePresent(PT, reference_string[i])))
      {
         page_fault++;
         if(flag==false && current < no_of_frames)
         {

            frame[current]=reference_string[i];
            printFrameContents(frame, no_of_frames);
            updatePageTable(PT, reference_string[i], current, true, false);
            current = current + 1;
            if(current == no_of_frames)
            {
                flag=true;
            }

         }

         else // Frame is full, apply Second Chance Algorithm
         {
            searchSecondChancePage(PT, frame, no_of_frames, &sc_page_index);
            updatePageTable(PT, frame[sc_page_index], -1, false, false);
            frame[sc_page_index]=reference_string[i];
            printFrameContents(frame, no_of_frames);
            updatePageTable(PT, reference_string[i], sc_page_index, true, false);
         }
      }
      PT[reference_string[i]].second_chance = true;
    }


   printf("\nTotal Number of Page Faults = %d\n", page_fault);
   printf("\nPage Fault ratio = %.2f\n", (float)page_fault/n);
   printf("\nPage Hit Ratio = %.2f\n", (float)(n- page_fault)/n);
   return 0;
}
```

Output:



```
Enter the number of pages:
20

Enter the reference string (different page numbers):
7 1 0 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter the number of frames you want to allocate to the process:3

****The Contents inside the Frame array at different times:****
7 -1 -1
7 1 -1
7 1 0
2 1 0
2 3 0
4 3 0
4 2 0
4 2 3
0 2 3
1 2 3
1 2 0
1 7 0

Total Number of Page Faults = 12

Page Fault ratio = 0.60

Page Hit Ratio = 0.40
```