Week - 9
(17 August 2023)
Experiment - 8

Question:

(i) Write a C program to simulate page replacement algorithms:

(a) FIFO

(b) LRU

(c) Optimal

(ii) Write a C program to simulate the following file allocation strategies:

(a) Sequential

(b) Indexed

(c) Linked

Program:

(i) Page Replacement Algorithms:

```
#include<stdio.h>
int n, nf, i, j, k;
int in[100];
int p[50];
int hit=0;
int pgfaultcnt=0;

void getData()
{
   printf("\nEnter length of page reference sequence:");
   scanf("%d",&n);
   printf("\nEnter the page reference sequence:");
   for(i=0; i<n; i++)
      scanf("%d",&in[i]);
   printf("\nEnter no of frames:");
   scanf("%d",&nf);
}

void initialize()
{
   pgfaultcnt=0;
   for(i=0; i<nf; i++)
      p[i]=9999;
}

int isHit(int data)
{
   hit=0;
   for(j=0; j<nf; j++)
```

```
    {
        if(p[j]==data)
        {
            hit=1;
            break;
        }
    }
    return hit;
}

int getHitIndex(int data)
{
    int hitind;
    for(k=0; k<nf; k++)
    {
        if(p[k]==data)
        {
            hitind=k;
            break;
        }
    }
    return hitind;
}

void dispPages()
{
    for (k=0; k<nf; k++)
    {
        if(p[k]!=9999)
            printf(" %d",p[k]);
    }
}

void dispPgFaultCnt()
{
    printf("\nTotal no of page faults:%d",pgfaultcnt);
}

void fifo()
{
    initialize();
    for(i=0; i<n; i++)
    {
        printf("\nFor %d :",in[i]);

        if(isHit(in[i])==0)
        {
```

```
            for(k=0; k<nf-1; k++)
               p[k]=p[k+1];

            p[k]=in[i];
            pgfaultcnt++;
            dispPages();
         }
         else
            printf("No page fault");
      }
      dispPgFaultCnt();
}


void optimal()
{
      initialize();
      int near[50];
      for(i=0; i<n; i++)
      {

         printf("\nFor %d :",in[i]);

         if(isHit(in[i])==0)
         {

            for(j=0; j<nf; j++)
            {
               int pg=p[j];
               int found=0;
               for(k=i; k<n; k++)
               {
                  if(pg==in[k])
                  {
                     near[j]=k;
                     found=1;
                     break;
                  }
                  else
                     found=0;
               }
               if(!found)
                  near[j]=9999;
            }
            int max=-9999;
            int repindex;
```

```
            for(j=0; j<nf; j++)
            {
               if(near[j]>max)
               {
                  max=near[j];
                  repindex=j;
               }
            }
            p[repindex]=in[i];
            pgfaultcnt++;

            dispPages();
         }
         else
            printf("No page fault");
      }
      dispPgFaultCnt();
}

void lru()
{
   initialize();

   int least[50];
   for(i=0; i<n; i++)
   {

      printf("\nFor %d :",in[i]);

      if(isHit(in[i])==0)
      {

         for(j=0; j<nf; j++)
         {
            int pg=p[j];
            int found=0;
            for(k=i-1; k>=0; k--)
            {
               if(pg==in[k])
               {
                  least[j]=k;
                  found=1;
                  break;
               }
               else
                  found=0;
            }
```

```
            if(!found)
               least[j]=-9999;
         }
         int min=9999;
         int repindex;
         for(j=0; j<nf; j++)
         {
            if(least[j]<min)
            {
               min=least[j];
               repindex=j;
            }
         }
         p[repindex]=in[i];
         pgfaultcnt++;

         dispPages();
      }
      else
         printf("No page fault!");
   }
   dispPgFaultCnt();
}

int main()
{
   int choice;
   while(1)
   {
      printf("\nPage Replacement Algorithms\n1.Enter data\n2.FIFO\n3.Optimal\n4.LRU\n5.Exit\nEnter your choice:");
      scanf("%d",&choice);
      switch(choice)
      {
      case 1: getData();
            break;
      case 2: fifo();
            break;
      case 3: optimal();
            break;
      case 4: lru();
            break;
      default: return 0;
             break;
      }
   }
}
```

Output: (Page Replacement Algorithms):

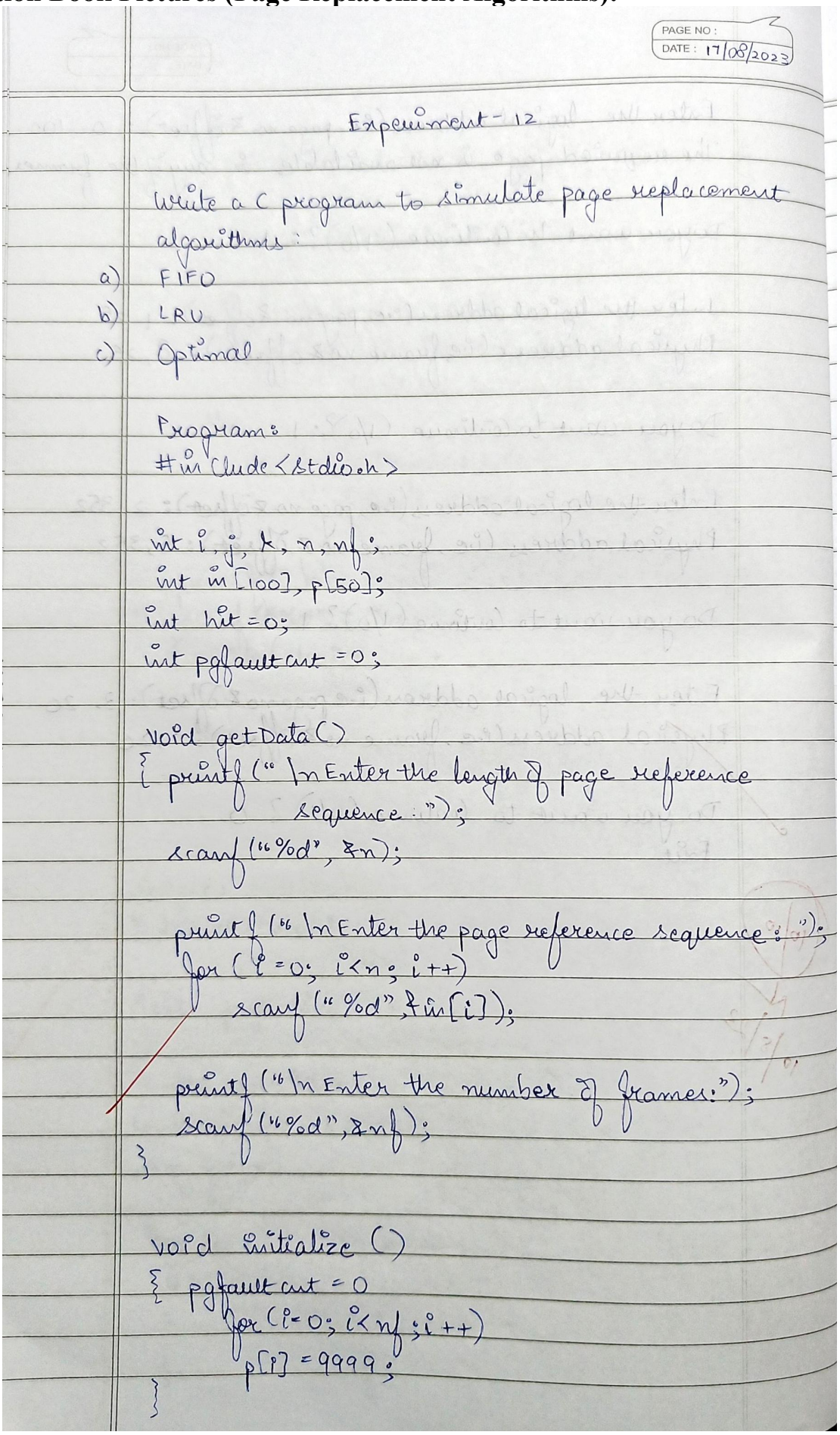
```
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:1

Enter length of page reference sequence:8

Enter the page reference sequence:2 3 4 2 3 5 6 2

Enter no of frames:3
```

```
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:2

For 2 : 2
For 3 : 2 3
For 4 : 2 3 4
For 2 :No page fault
For 3 :No page fault
For 5 : 3 4 5
For 6 : 4 5 6
For 2 : 5 6 2
Total no of page faults:6
```

```
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:3

For 2 : 2
For 3 : 2 3
For 4 : 2 3 4
For 2 :No page fault
For 3 :No page fault
For 5 : 2 5 4
For 6 : 2 6 4
For 2 :No page fault
Total no of page faults:5
```

```
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:4

For 2 : 2
For 3 : 2 3
For 4 : 2 3 4
For 2 :No page fault!
For 3 :No page fault!
For 5 : 2 3 5
For 6 : 6 3 5
For 2 : 6 2 5
Total no of page faults:6
```

Observation Book Pictures (Page Replacement Algorithms):



PAGE NO :
DATE : 17/08/2023

Experiment - 12

Write a C program to simulate page replacement algorithms:

a) FIFO
b) LRU
c) Optimal

Program:

```
#include<stdio.h>

int i, j, k, n, nf;
int in[100], p[50];
int hit=0;
int pgfaultcnt=0;

void getData()
{ printf("\n Enter the length of page reference
          sequence:");
  scanf("%d", &n);

  printf("\n Enter the page reference sequence:");
  for(i=0; i<n; i++)
    scanf("%d", &in[i]);

  printf("\n Enter the number of frames:");
  scanf("%d", &nf);
}

void initialize()
{ pgfaultcnt=0
    for(i=0; i<nf; i++)
      p[i]=9999;
}
```

```
int isHit(int data)
{ hit = 0;
  for(j=0; j<nf; j++)
  { if(p[j] == data)
    { hit = 1;
      break;
    }
  }
  return hit;
}

int getHitIndex(int data)
{ int hitind;
  for(k=0; k<nf; k++)
  { if(p[k] == data)
    { hitind = k;
      break;
    }
  }
  return hitind;
}

void dispPages()
{ for(k=0; k<nf; k++)
  { if(p[k] != 9999)
      printf("%d", p[k]);
  }
}

void dispPgFaultCnt()
{ printf("\nTotal no. of page faults: %d", pgfaultcnt);
}
```

→

```
void fifo()
{ initialize();
  for(i=0; i<n; i++)
  { printf("\n For %od: ", in[i]);
    if(isHit(in[i])==0)
    { for(k=0; k<nf-1; k++)
        p[k] = p[k+1];

      p[k] = in[i];
      pgfaultcnt++;
      dispPages();
    }
  else
    printf("No page fault");
  }
  dispPgFaultCnt();
}

void optimal()
{ initialize();
  int near[50];
  for(i=0; i<n; i++)
  { printf("\n For %od: ", in[i]);

    if(isHit(in[i])==0)
    { for(j=0; j<nf; j++)
      { int pg = p[j];
        int found = 0;
        for(k=i; k<n; k++)
        { if(pg==in[k])
          { near[j] = k;
            found = 1;
            break;
          }
```

```
            else
                found = 0;
        }
        if (! found)
            near[j] = 9999;
    }
    int max = -9999;
    int repindex;
    for (j=0; j<nf; j++)
    {   if (near[j] > max)
        {   max = near[j];
            repindex = j;
        }
    }

    p[repindex] = in[i];
    pgfaultcnt++;
    dispPages();
    }
    else
        printf("No page fault");
    }
    dispPgFaultCnt();
}

void lru()
{ initialize();
  int least[50];
  for (i=0; i<n; i++)
  {   printf("\n For %d: ", in[i]);
      if (isHit(in[i]) == 0)
      {   for (j=0; j<nf; j++)
          {
```

```
int pg = p[j];
int found = 0;
for(k = i-1; k>0; k--)
{ if (pg == in[k])
  { least[j] = k;
    found = 1;
    break;
  }
else
  { found = 0;
  }
if (!found)
  least[j] = -9999;
}

int min = 9999;
int repindex;
for(j=0; j<nf; j++)
{ if(least[j] < min)
  { min = least[j];
    repindex = j;
  }
}

p[repindex] = in[i];
pgfaultcnt++;
dispPages();
}
else
  printf("No page fault!");
}
dispPgFaultCnt();
}
```

```
int main()
{ int choice;
  while(1)
  { printf("Page Replacement Algorithms:\n
        1. Enter data\n2. FIFO\n3. Optimal\n
        4. LRU\n5. Exit\n\nEnter your choice:");
    scanf("%d", &choice);

    switch(choice)
    { case 1: getData()
              break;

      case 2: fifo()
              break;

      case 3: Optimal()
              break;

      case 4: lru()
              break;

      default: return 0;
               break;
    }
  }
}
```

Output:

Page Replacement Algorithms:

1. Enter Data
2. FIFO
3. Optimal
4. LRU
5. Exit

Enter your choice: 1

Enter length of page reference sequence: 8

Enter the page reference sequence: 2 3 4 2 3 5 6 2

Enter the number of frames: 3

Enter your choice: 2

FIFO:

For 2: 2

For 3: 2 3

For 4: 2 3 4

For 2: No page fault

For 3: No page fault

For 5: 3 4 5

For 6: 4 5 6

For 2: 5 6 2

Total no. of page faults: 6

Enter your choice: 3

Optimal:

For 2: 2

For 3: 2 3

PAGE NO :
DATE :

For 4 : 2 3 4
For 2 : No page fault
For 3 : No page fault
For 5 : 2 5 4
For 6 : 2 6 4
For 2 : No page fault.

Total no. of page faults : 5

Enter your choice : 4
LRU :
For 2 : 2
For 3 : 2 3
For 4 : 2 3 4
For 2 : No page fault
For 3 : No page fault
For 5 : 2 3 5
For 6 : 6 3 5
For 2 : 6 2 5

Total no. of page faults : 6

10/10

17/8/23

(ii) File Allocation Strategies:

(a) <u>Sequential</u>:

```
#include<stdio.h>
#include<string.h>

struct fileTable
{ char name[20];
int sb, nob; }
ft[30];

void main() {
   int i, j, n; char s[20];
   printf("Enter no of files  :");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      printf("\nEnter file name %d  :",i+1);
      scanf("%s",ft[i].name);
      printf("Enter starting block of file %d  :",i+1);
      scanf("%d",&ft[i].sb);
      printf("Enter no of blocks in file %d :",i+1);
      scanf("%d",&ft[i].nob);
   }
   printf("\nEnter the file name to be searched -- ");
   scanf("%s",s);
   for(i=0;i<n;i++)
   if(strcmp(s, ft[i].name)==0)
   break;
   if(i==n)
   printf("\nFile Not Found");
   else
   {
      printf("\nFILE  NAME      START  BLOCK      NO  OF  BLOCKS      BLOCKS
OCCUPIED\n");
      printf("\n%s\t\t%d\t\t%d\t",ft[i].name,ft[i].sb,ft[i].nob);
      for(j=0;j<ft[i].nob;j++)
      printf("%d, ",ft[i].sb+j);
   }
}
```

(b) Indexed:

```
#include<stdio.h>
#include<conio.h>

struct fileTable
{
   char name[20];
   int nob, blocks[30];
}ft[30];

void main()
{
   int i, j, n; char s[20];
   printf("Enter no of files  :");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      printf("\nEnter file name %d  :",i+1);
      scanf("%s",ft[i].name);
      printf("Enter no of blocks in file %d :",i+1);
      scanf("%d",&ft[i].nob);
      printf("Enter the blocks of the file    :");
          for(j=0;j<ft[i].nob;j++)
              scanf("%d",&ft[i].blocks[j]);
   }

   printf("\nEnter the file name to be searched -- ");
   scanf("%s",s);  for(i=0;i<n;i++)

   if(strcmp(s, ft[i].name)==0)
   break;

   if(i==n)
      printf("\nFile Not Found");

   else
   {
      printf("\nFILE NAME  NO OF BLOCKS   BLOCKS OCCUPIED");
      printf("\n    %s\t\t%d\t",ft[i].name,ft[i].nob);
      for(j=0;j<ft[i].nob;j++)
         printf("%d, ",ft[i].blocks[j]);
   }
}
```

(c) Linked:

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

struct fileTable
{
   char name[20];
   int nob;
   struct block *sb;
}ft[30];

struct block
{
   int bno;
   struct block *next;
};

void main()
{
   int i, j, n;
   char s[20];
   struct block *temp;
   printf("Enter no of files  :");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      printf("\nEnter file name %d  :",i+1);
      scanf("%s",ft[i].name);
      printf("Enter no of blocks in file %d :",i+1);
      scanf("%d",&ft[i].nob);

      ft[i].sb=(struct block*)malloc(sizeof(struct block));
      temp = ft[i].sb;

      printf("Enter the blocks of the file    :");
      scanf("%d",&temp->bno);

      temp->next=NULL;

      for(j=1;j<ft[i].nob;j++)
      {
         temp->next = (struct block*)malloc(sizeof(struct block));
         temp = temp->next;
         scanf("%d",&temp->bno);
      }
```

```
        temp->next = NULL;
    }
        printf("\nEnter the file name to be searched   --   ");
        scanf("%s",s);
        for(i=0;i<n;i++)
            if(strcmp(s, ft[i].name)==0)
        break;

        if(i==n)
        printf("\nFile Not Found");

        else
        {
            printf("\nFILE NAME  NO OF BLOCKS   BLOCKS OCCUPIED");
            printf("\n    %s\t\t%d\t",ft[i].name,ft[i].nob);
            temp=ft[i].sb;
            for(j=0;j<ft[i].nob;j++)
            {
                printf("%d->",temp->bno);
                temp = temp->next;
            }
        }
}
```

(iii) Output (File Allocation Strategies):

(a) Sequential:

```
Enter no of files  :3

Enter file name 1  :A
Enter starting block of file 1  :85
Enter no of blocks in file 1 :6

Enter file name 2  :B
Enter starting block of file 2  :102
Enter no of blocks in file 2 :4

Enter file name 3  :C
Enter starting block of file 3  :60
Enter no of blocks in file 3 :4

Enter the file name to be searched -- B

FILE NAME   START BLOCK   NO OF BLOCKS   BLOCKS OCCUPIED

B               102             4       102, 103, 104, 105,
```

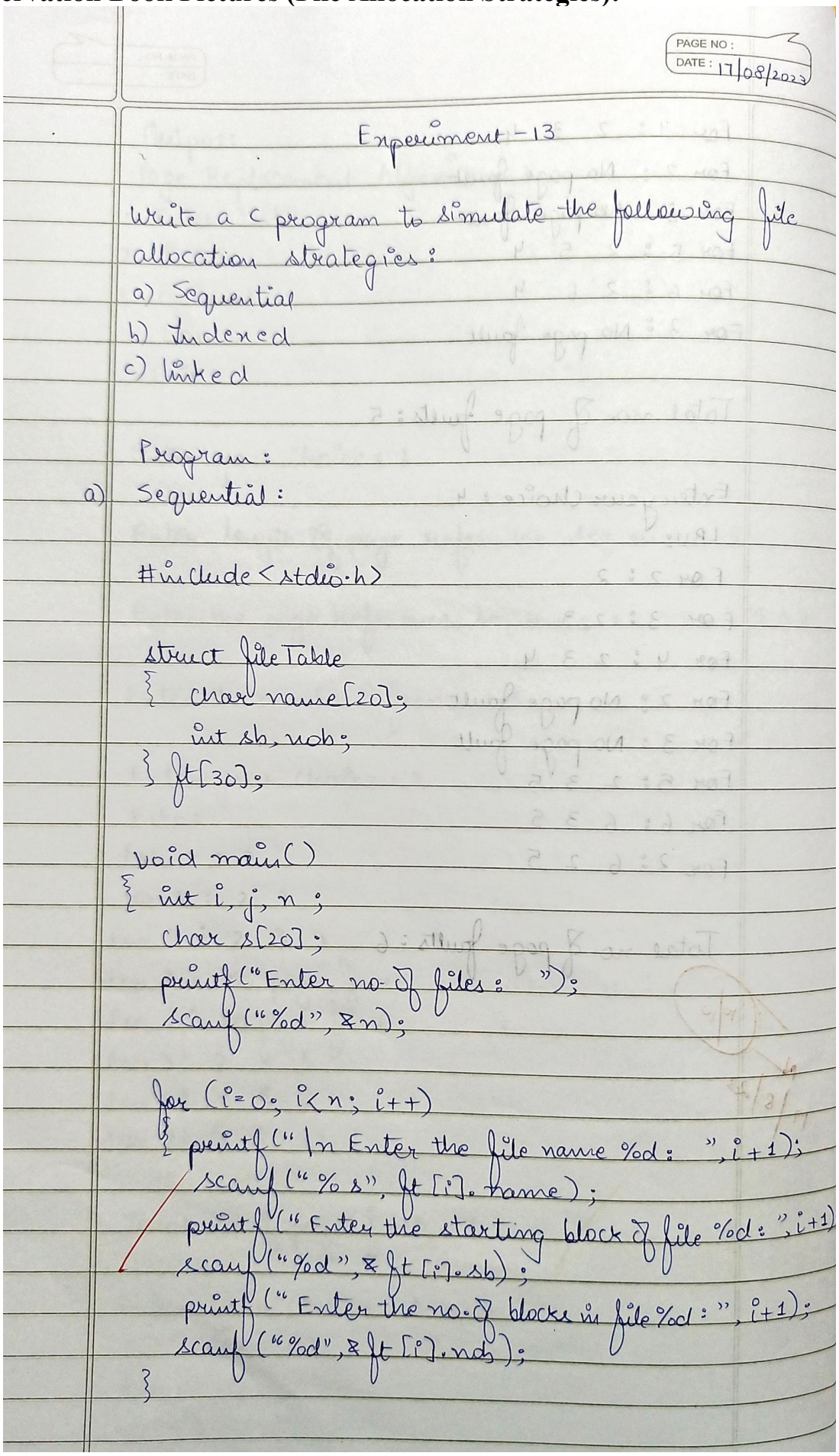
(b) Indexed:

```
Enter no of files  :2

Enter file name 1  :A
Enter no of blocks in file 1 :4
Enter the blocks of the file    :12 23 9 4

Enter file name 2  :G
Enter no of blocks in file 2 :5
Enter the blocks of the file    :88 77 66 55 44

Enter the file name to be searched -- G

FILE NAME  NO OF BLOCKS   BLOCKS OCCUPIED
    G           5       88, 77, 66, 55, 44,
```

(c) Linked:

```
Enter no of files  :2

Enter file name 1  :A
Enter no of blocks in file 1 :4
Enter the blocks of the file    :12 23 9 4

Enter file name 2  :G
Enter no of blocks in file 2 :5
Enter the blocks of the file    :88 77 66 55 44

Enter the file name to be searched   --   G

FILE NAME  NO OF BLOCKS   BLOCKS OCCUPIED
    G           5       88->77->66->55->44->
```

1.1.1 Observation Book Pictures (File Allocation Strategies):



PAGE NO :
DATE : 17/08/2023

Experiment - 13

Write a C program to simulate the following file allocation strategies:
a) Sequential
b) Indexed
c) Linked

Program:

a) Sequential:

```
#include<stdio.h>

struct fileTable
{ char name[20];
  int sb, nob;
} ft[30];

void main()
{ int i, j, n;
  char s[20];
  printf("Enter no. of files: ");
  scanf("%d", &n);

  for (i=0; i<n; i++)
  { printf("\n Enter the file name %d: ", i+1);
    scanf("%s", ft[i].name);
    printf("Enter the starting block of file %d: ", i+1);
    scanf("%d", &ft[i].sb);
    printf("Enter the no. of blocks in file %d: ", i+1);
    scanf("%d", &ft[i].nob);
  }
```

```
printf("\n Enter the file name to be searched : ");
scanf("%s", s);
for(i=0; i<n; i++)
    if (strcmp(s, ft[i].name)==0)
        break;
    if (i==n)
        printf("\n File Not Found");

    else
    { printf("\n FILE NAME  START BLOCK  NO OF BLOCKS
                BLOCKS OCCUPIED\n");
      printf("\n %s\t\t %d\t\t %d\t", ft[i].name,
              ft[i].sb, ft[i].nob);
      for(j=0; j<ft[i].nob; j++)
          printf(" %d", ft[i].sb+j);
    }
}
```

Output:

Enter no. of files: 3

Enter file name 1: A

Enter starting block of file 1: 85

Enter the no. of blocks in file 1: 6

Enter file name 2: B

Enter starting block of file 2: 102

Enter the no. of blocks in file 2: 4

Enter file name 3: C

Enter starting block of file 3: 60

Enter the no. of blocks in file 3: 4

Enter the file name to be searched: B.

FILE NAME	START BLOCK	NO OF BLOCKS	BLOCKS OCCUPIED
B	102	4	102, 103, 104, 105

b) Indexed:

```
#include<stdio.h>

struct fileTable
{ char name[20];
  int nob, blocks[30];
} ft[30];

void main()
{ int i, j, n;
  char s[20];
  printf("Enter no. of files: ");
  scanf("%d", &n);
  for(i=0; i<n; i++)
  { printf("Enter the file name %d: ", i+1);
    scanf("%s", ft[i].name);
    printf("Enter no. of blocks in file %d: ", i+1);
    scanf("%d", &ft[i].nob);
    printf("Enter the blocks of the file: ");
    for(j=0; j<ft[i].nob; j++)
      scanf("%d", &ft[i].blocks[j]);
  }

  printf("\n Enter the file name to be searched: ");
  scanf("%s", s);
  for(i=0; i<n; i++)
    if(strcmp(s, ft[i].name)==0)
      break;
  if(i==n)
    printf("\n File not found");
```

→

```
else
{ printf("\n FILE NAME  NO. OF BLOCKS
                BLOCKS OCCUPIED");
  printf("\n %s \t\t %d \t", ft[i].name,
                ft[i].nob);
  for (j=0; j<ft[i].nob; j++)
      printf("%d", ft[i].blocks[j]);
}
}
```

Output:

Enter no. of files : 2

Enter File 1 name : A

Enter no. of blocks in File 1 : 4

Enter the blocks of File 1 : 12 23 9 4

Enter File 2 name : G

Enter no. of blocks in File 2 : 5

Enter the blocks of File 2 : 88 77 66 55 44

Enter File to be searched : G

FILE NAME	No. OF BLOCKS	BLOCKS OCCUPIED
G	5	88, 77, 66, 55, 44.

→

c) Linked:

```
#include<stdio.h>

struct FileTable
{ char name[20];
  int nob;
  struct blocks *sb;
} ft[30];

struct block
{ int bno;
  struct block *next;
};

Void main()
{ int i, j, n;
  char s[20];
  struct block *temp;
  printf("Enter no. of files: ");
  scanf("%d", &n);
  for(i=0; i<n; i++)
  { printf("\nEnter the filename %d: " i+1);
    scanf("%s", ft[i].name);
    printf("Enter no. of blocks in File %d: ", i+1);
    scanf("%d", &ft[i].nob);
    ft[i].sb = (struct block *) malloc(sizeof(struct block));
    temp = ft[i].sb;
    printf("Enter the blocks of the file: ");
    scanf("%d", &temp->bno);
    temp->next = NULL;

→
```

```
for(j=1; j<ft[i].nob; j++)
{ temp->next = (struct block*) malloc(sizeof(struct block));
  temp = temp->next;
  scanf("%d", &temp->bno);
}
temp->next = NULL;
}
printf("\n Enter the filename to be searched: ");
scanf("%s", s);
for(i=0; i<n; i++)
  if(strcmp(s, ft[i].name) == 0)
    break;
if(i==n)
  printf("\n File not found");

else
{ printf("\n FILE NAME  NO OF BLOCKS  BLOCKS OCCUPIED");
  printf("\n %s \t\t %d \t ", ft[i].name, ft[i].nob);
  temp = ft[i].sb;
  for(j=0; j<ft[i].nob; j++)
  { printf("%d ->", temp->bno);
    temp = temp->next;
  }
}
```

Output:

Enter the no. of files : 2

Enter file name 1 : A

Enter no. of blocks in file 1 : 4

Enter the blocks of the file 1 : 12 23 9 4

PAGE NO :
DATE :

Enter file name 2 : G
Enter no. of blocks in file 2 : 5
Enter the blocks of file 2 : 88 77 66 55 44

Enter the file to be searched : G

FILENAME	NO. OF BLOCKS	BLOCKS OCCUPIED
G	5	88 → 77 → 66 → 55 → 44