```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


int main() {

    int choice;

    char msg[1000], old_name[20], new_name[20], source_file[20], target_file[20];

    FILE *file, *source, *target;


    printf("\n1. Create File and Write data\n2. Read the data\n3. Rename File\n4. Copy Data of File to
another\nEnter The Choice: ");

    scanf("%d", &choice);


    switch (choice) {

        case 1:

            file = fopen("program2.txt", "w");

            if (file == NULL) {

                printf("Error opening file!");

                exit(1);

            }

            printf("Enter message: ");

            fgets(msg, sizeof(msg), stdin);

            fprintf(file, "%s", msg);

            fclose(file);

            break;

        case 2:

            file = fopen("program2.txt", "r");

            if (file == NULL) {

                printf("Error opening file!");

                exit(1);

            }
            fscanf(file, "%[^\n]", msg);

            printf("Message Is: %s", msg);

            fclose(file);

            break;

        case 3:

            printf("\nEnter old file name: ");

            scanf("%s", old_name);

            printf("\nEnter new file name: ");

            scanf("%s", new_name);

            if (rename(old_name, new_name) == 0) {

                printf("File renamed successfully.\n");

            } else {

                printf("Unable to rename file. Please check if the file exists and you have permission to
modify it.\n");

            }

            break;

        case 4:

            printf("Enter name of file to copy: ");

            scanf("%s", source_file);

            source = fopen(source_file, "r");

            if (source == NULL) {

                printf("Unable to open source file. Exiting.\n");

                exit(EXIT_FAILURE);

            }

            printf("Enter name of target file: ");

            scanf("%s", target_file);

            target = fopen(target_file, "w");

            if (target == NULL) {

                printf("Unable to open target file. Exiting.\n");

                exit(EXIT_FAILURE);

            }
```

```
            while ((choice = fgetc(source)) != EOF) {
                fputc(choice, target);
            }
            printf("File copied successfully.\n");
            fclose(source);
            fclose(target);
            break;
        default:
            printf("Invalid choice.\n");
            break;
    }
    return 0;


}

//Program 2

#include <stdio.h>
#define MAX 100

int main() {
    int Arrival_time[MAX], Burst_time[MAX], Completion_time[MAX],
        Turn_Around_time[MAX], Waiting_time[MAX],
        Average_Turn_Around_time = 0, Average_Waiting_time = 0, i, j;

    printf("Enter the number of processes: ");
    scanf("%d", &j);

    // Input arrival time and burst time for each process
    for (i = 0; i < j; i++) {
```

```
    printf("Enter Arrival Time for Process %d: ", i + 1);
    scanf("%d", &Arrival_time[i]);
    printf("Enter Burst Time for Process %d: ", i + 1);
    scanf("%d", &Burst_time[i]);
}

// Calculate completion time for each process
Completion_time[0] = Burst_time[0];
for (i = 1; i < j; i++) {
    Completion_time[i] = Completion_time[i - 1] + Burst_time[i];
}

// Calculate turn around time and waiting time for each process
for (i = 0; i < j; i++) {
    Turn_Around_time[i] = Completion_time[i] - Arrival_time[i];
    Waiting_time[i] = Turn_Around_time[i] - Burst_time[i];
    Average_Waiting_time += Waiting_time[i];
    Average_Turn_Around_time += Turn_Around_time[i];
}

// Print table header
printf("\nProcess\tArrival(T)\tBurst(T)\tCompletion(T)\tTurn-Around(T)\tWaiting(T)");

// Print details for each process
for (i = 0; i < j; i++) {
    printf("\nP[%d]\t%d\t\t%d\t\t%d\t\t%d\t\t%d",
           i + 1, Arrival_time[i], Burst_time[i], Completion_time[i],
           Turn_Around_time[i], Waiting_time[i]);
}

// Calculate and print average turn around time and average waiting time
```

```
    printf("\n\nAverage Turn Around Time: %.2f", (float)Average_Turn_Around_time / j);

    printf("\nAverage Waiting Time: %.2f\n", (float)Average_Waiting_time / j);


    return 0;

}
```

```
// Practical -: 03 TITLE: - Producer-Consumer problem using
semaphores
#include <stdio.h>
void main() {
int buffer[10], bufsize, in, out, produce, consume, choice = 0;
in = 0;
out = 0;
bufsize = 10;
while (choice != 3) {
printf("\n1. Produce \t2. Consume \t3. Exit");
printf("\nEnter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
if ((in + 1) % bufsize == out)
printf("\nBuffer is Full");
else {
printf("\nEnter the value: ");
scanf("%d", &produce);
buffer[in] = produce;
in = (in + 1) % bufsize;
}
break;
case 2:
if (in == out)
printf("\nBuffer is Empty");
else {
consume = buffer[out];
printf("\nThe consumed value is %d", consume);
out = (out + 1) % bufsize;
}
break;
}
}
}
/* void main(){
int buffer[10], bufsize, in, out, produce, consume, choice=0;
int = 0;
out = 0;
buffsize = 10;
while (choice !=3){
```

```
printf("\n1. Produce \t2. Consume \t3. Exit");
printf("\nEnter your choice: ");
scanf("%d", &choice);
}
}
*/


//program 4

#include <stdio.h>

#define max 25


int main() {

  int frag[max], b[max], f[max], i, j, nb, nf, temp;

  static int bf[max] = {0}, ff[max] = {0}; // Initialize arrays


  printf("\n\tMemory Management Scheme - First Fit");

  printf("\nEnter the number of blocks: ");

  scanf("%d", &nb);

  printf("Enter the number of files: ");

  scanf("%d", &nf);


  printf("\nEnter the size of the blocks:\n");

  for (i = 1; i <= nb; i++) {

    printf("Block %d: ", i);

    scanf("%d", &b[i]);

  }


  printf("Enter the size of the files:\n");

  for (i = 1; i <= nf; i++) {

    printf("File %d: ", i);

    scanf("%d", &f[i]);

  }
```

```c
    for (i = 1; i <= nf; i++) {
        for (j = 1; j <= nb; j++) {
            if (bf[j] != 1) {
                temp = b[j] - f[i];
                if (temp >= 0) {
                    ff[i] = j;
                    break;
                }
            }
        }
        frag[i] = temp;
        bf[ff[i]] = 1;
    }

    printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment\n");
    for (i = 1; i <= nf; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i, f[i], ff[i], b[ff[i]], frag[i]);
    }

    return 0;
}
```

```c
// Practical -: 05 TITLE: - Sequential File allocation
strategies
#include <stdio.h>
#include <string.h>

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

int main() {
    int i, j, n;
    char s[20];

    printf("Enter the number 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 number 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) {
            printf("\nFILE NAME\tSTART BLOCK\tNO OF BLOCKS\tBLOCKS
OCCUPIED\n");
            printf("%s\t\t%d\t\t%d\t\t", ft[i].name, ft[i].sb, ft[i].nob);
            for (j = 0; j < ft[i].nob; j++) {
                printf("%d, ", ft[i].sb + j);
            }
            return 0;
        }
    }

    printf("\nFile Not Found");
    return 0;
}
//program 6

#include<stdio.h>

#include<string.h>

#include<stdlib.h>
```

```
struct Directory {
    char dname[10];
    char fname[10][10];
    int fcnt;
} dir;

void createFile() {
    if (dir.fcnt < 10) {
        printf("\nEnter the name of the file: ");
        scanf("%s", dir.fname[dir.fcnt]);
        dir.fcnt++;
    } else {
        printf("\nDirectory is full. Cannot create more files.");
    }
}

void deleteFile() {
    char f[30];
    int i;
    printf("\nEnter the name of the file: ");
    scanf("%s", f);
    for (i = 0; i < dir.fcnt; i++) {
        if (strcmp(f, dir.fname[i]) == 0) {
            printf("File %s is deleted\n", f);
            strcpy(dir.fname[i], dir.fname[dir.fcnt - 1]);
            dir.fcnt--;
            return;
        }
    }
    printf("File %s not found\n", f);
}
```

```
void searchFile() {
    char f[30];
    int i;
    printf("\nEnter the name of the file: ");
    scanf("%s", f);
    for (i = 0; i < dir.fcnt; i++) {
        if (strcmp(f, dir.fname[i]) == 0) {
            printf("File %s is found\n", f);
            return;
        }
    }
    printf("File %s not found\n", f);
}

void displayFiles() {
    int i;
    if (dir.fcnt == 0) {
        printf("\nDirectory is empty\n");
    } else {
        printf("\nThe Files are:");
        for (i = 0; i < dir.fcnt; i++) {
            printf("\t%s", dir.fname[i]);
        }
        printf("\n");
    }
}

int main() {
    int ch;
    dir.fcnt = 0;
    printf("\nEnter name of directory: ");
    scanf("%s", dir.dname);

    while (1) {
        printf("\n\n1. Create File\t2. Delete File\t3. Search File\n4. Display Files\t5. Exit\nEnter your
choice: ");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                createFile();
                break;
            case 2:
                deleteFile();
                break;
            case 3:
                searchFile();
                break;
            case 4:
                displayFiles();
                break;
            default:
                exit(0);
        }
    }

    return 0;
}

// Practical -: 07 TITLE: - FIFO page replacement algorithm
#include<stdio.h>
```

```
#define MAX_FRAMES 10

#define MAX_PAGES 25


int main() {

  int frames[MAX_FRAMES];

  int pages[MAX_PAGES];

  int n_frames, n_pages;

  int page_faults = 0;

  int frame_index = 0;


  printf("Enter the number of frames: ");

  scanf("%d", &n_frames);


  printf("Enter the number of pages: ");

  scanf("%d", &n_pages);


  printf("Enter the page reference string: ");

  for (int i = 0; i < n_pages; i++) {

    scanf("%d", &pages[i]);

  }


  // Initialize frames to -1, indicating empty

  for (int i = 0; i < n_frames; i++) {

    frames[i] = -1;

  }


  printf("\nPage Replacement Process (FIFO):\n");

  for (int i = 0; i < n_pages; i++) {

    int page = pages[i];

    int found = 0;
```

```
        // Check if page is already in memory
        for (int j = 0; j < n_frames; j++) {
            if (frames[j] == page) {
                found = 1;
                break;
            }
        }

        if (!found) {
            // Page fault: replace the oldest page
            printf("Page fault at page %d\n", page);
            frames[frame_index] = page;
            frame_index = (frame_index + 1) % n_frames;
            page_faults++;
        }

        // Display current state of frames
        printf("Frames: ");
        for (int j = 0; j < n_frames; j++) {
            printf("%d ", frames[j]);
        }
        printf("\n");
    }

    printf("\nTotal Page Faults: %d\n", page_faults);

    return 0;
}
//program 8 fcfs disk scheduinh
#include <stdio.h>
```

```
int main() {
   int t[20], n, i, tohm[20], tot = 0;
   float avhm;

   printf("Enter the number of tracks: ");
   scanf("%d", &n);

   printf("Enter the tracks to be traversed: ");
   for (i = 0; i < n; i++)
      scanf("%d", &t[i]);

   for (i = 0; i < n - 1; i++) {
      tohm[i] = t[i + 1] - t[i];
      if (tohm[i] < 0)
         tohm[i] = tohm[i] * (-1);
      tot += tohm[i];
   }

   avhm = (float)tot / n;

   printf("\nTracks traversed\tDifference between tracks\n");
   for (i = 0; i < n - 1; i++)
      printf("%d\t\t\t%d\n", t[i], tohm[i]);

   printf("\nAverage header movements: %f\n", avhm);

   return 0;
}
```