

19)

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

```
#include <pthread.h>
```

```
#define NUM_THREADS 5
```

```
pthread_mutex_t lock;
```

```
void* threadFunction(void* arg) {
```

```
    pthread_mutex_lock(&lock);
```

```
    printf("Thread %d is in the critical section.\n", *(int*)arg);
```

```
    pthread_mutex_unlock(&lock);
```

```
    return NULL;
```

```
}
```

```
int main() {
```

```
    pthread_t threads[NUM_THREADS];
```

```
    int threadIds[NUM_THREADS];
```

```
    pthread_mutex_init(&lock, NULL);
```

```
    for (int i = 0; i < NUM_THREADS; i++) {
```

```
        threadIds[i] = i;
```

```
        pthread_create(&threads[i], NULL, threadFunction, &threadIds[i]);
```

```
    }
```

```
    for (int i = 0; i < NUM_THREADS; i++) {
```

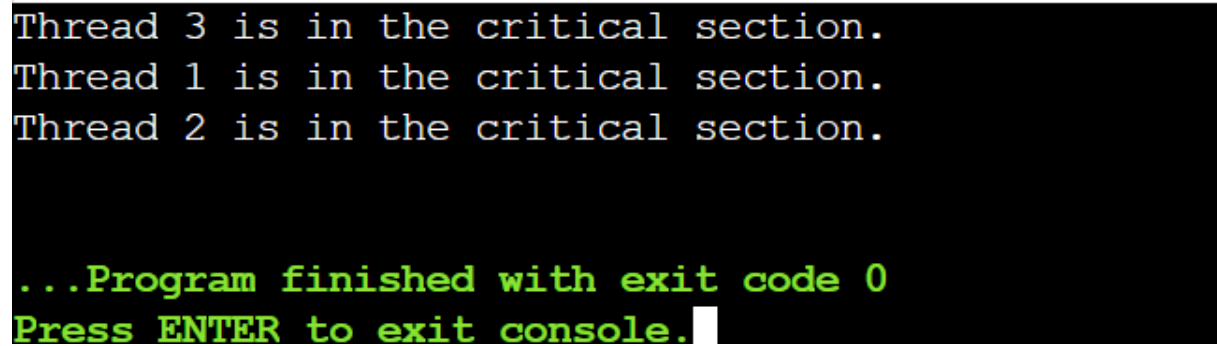
```
        pthread_join(threads[i], NULL);
```

```
    }
```

```
    pthread_mutex_destroy(&lock);
```

```
    return 0;
}
```

OUTPUT:

A screenshot of a terminal window with a black background and white and green text. The output shows three lines of status messages for threads, followed by a green message indicating the program finished with exit code 0 and a prompt to press ENTER to exit the console.

```
Thread 3 is in the critical section.
Thread 1 is in the critical section.
Thread 2 is in the critical section.

...Program finished with exit code 0
Press ENTER to exit console.
```

20)

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define READERS 5
#define WRITERS 2
sem_t mutex, writeBlock;
int readCount = 0;
void* reader(void* arg) {
    int id = *(int*)arg;
    while (1) {
        sem_wait(&mutex);
        readCount++;
        if (readCount == 1) sem_wait(&writeBlock);
        sem_post(&mutex);
        printf("Reader %d is reading.\n", id);
        sleep(1);
        sem_wait(&mutex);
        readCount--;
```

```

        if (readCount == 0) sem_post(&writeBlock);

        sem_post(&mutex);

        sleep(1);
    }

    return NULL;
}

void* writer(void* arg) {
    int id = *(int*)arg;

    while (1) {
        sem_wait(&writeBlock);

        printf("Writer %d is writing.\n", id);

        sleep(2);

        sem_post(&writeBlock);

        sleep(1);
    }

    return NULL;
}

int main() {
    pthread_t r[READERS], w[WRITERS];

    int ids[READERS + WRITERS];

    sem_init(&mutex, 0, 1);

    sem_init(&writeBlock, 0, 1);

    for (int i = 0; i < READERS; i++) {
        ids[i] = i + 1;

        pthread_create(&r[i], NULL, reader, &ids[i]);
    }

    for (int i = 0; i < WRITERS; i++) {
        ids[READERS + i] = i + 1;

        pthread_create(&w[i], NULL, writer, &ids[READERS + i]);
    }

    for (int i = 0; i < READERS; i++) pthread_join(r[i], NULL);
}

```

```

    for (int i = 0; i < WRITERS; i++) pthread_join(w[i], NULL);

    sem_destroy(&mutex);

    sem_destroy(&writeBlock);

    return 0;
}

```

**OUT PUT:**

```

Reader 1 is reading.
Reader 3 is reading.
Reader 2 is reading.
Reader 4 is reading.
Reader 5 is reading.
Writer 1 is writing.
Writer 2 is writing.
Reader 1 is reading.
Reader 3 is reading.
Reader 2 is reading.
Reader 4 is reading.
Reader 5 is reading.

```

21)

```

#include <stdio.h>

int main() {

    int blockSize[] = {100, 500, 200, 300, 600};

    int processSize[] = {212, 417, 112, 426};

    int n = sizeof(blockSize) / sizeof(blockSize[0]);

    int m = sizeof(processSize) / sizeof(processSize[0]);

    int allocation[m];

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

        int worstIdx = -1;

        for (int j = 0; j < n; j++) {

            if (blockSize[j] >= processSize[i]) {

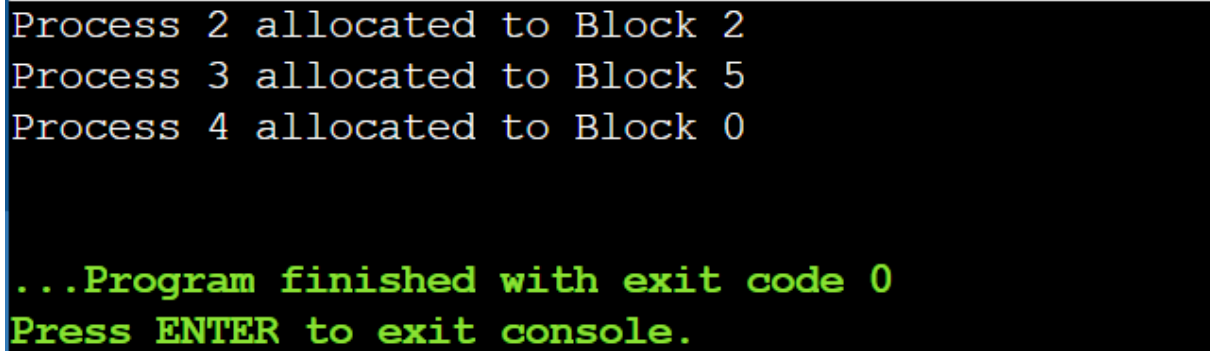
```

```

        if (worstIdx == -1 || blockSize[j] > blockSize[worstIdx])
            worstIdx = j;
    }
}
if (worstIdx != -1) {
    allocation[i] = worstIdx;
    blockSize[worstIdx] -= processSize[i];
} else {
    allocation[i] = -1;
}
}
for (int i = 0; i < m; i++)
    printf("Process %d allocated to Block %d\n", i + 1, allocation[i] + 1);
return 0;
}

```

OUT PUT:



```

Process 2 allocated to Block 2
Process 3 allocated to Block 5
Process 4 allocated to Block 0

...Program finished with exit code 0
Press ENTER to exit console.

```

22)

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

```
int main() {
```

```
    int blockSize[] = {100, 500, 200, 300, 600};
```

```

int processSize[] = {212, 417, 112, 426};

int n = sizeof(processSize) / sizeof(processSize[0]);

int m = sizeof(blockSize) / sizeof(blockSize[0]);

int allocation[n];

for (int i = 0; i < n; i++) {
    int bestIdx = -1;

    for (int j = 0; j < m; j++) {
        if (blockSize[j] >= processSize[i]) {
            if (bestIdx == -1 || blockSize[bestIdx] > blockSize[j])
                bestIdx = j;
        }
    }

    if (bestIdx != -1) {
        allocation[i] = bestIdx;
        blockSize[bestIdx] -= processSize[i];
    } else {
        allocation[i] = -1;
    }
}

printf("Process No.\tProcess Size\tBlock No.\n");

for (int i = 0; i < n; i++) {
    printf("%d\t\t%d\t\t", i + 1, processSize[i]);

    if (allocation[i] != -1)
        printf("%d\n", allocation[i] + 1);
    else
        printf("Not Allocated\n");
}

return 0;
}

```

**OUTPUT:**

Process No.	Process Size	Block No.
1	212	4
2	417	2
3	112	3
4	426	5

...Program finished with exit code 0  
 Press ENTER to exit console.

23)

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

```
int main() {
```

```
    int blockSize[] = {100, 500, 200, 300, 600};
```

```
    int processSize[] = {212, 417, 112, 426};
```

```
    int n = sizeof(processSize) / sizeof(processSize[0]);
```

```
    int m = sizeof(blockSize) / sizeof(blockSize[0]);
```

```
    int allocation[n];
```

```
    for (int i = 0; i < n; i++) {
```

```
        allocation[i] = -1;
```

```
        for (int j = 0; j < m; j++) {
```

```
            if (blockSize[j] >= processSize[i]) {
```

```
                allocation[i] = j;
```

```
                blockSize[j] -= processSize[i];
```

```
                break;
```

```
            }
```

```
        }
```

```
    }
```

```
    printf("Process No.\tBlock No.\n");
```

```
    for (int i = 0; i < n; i++)
```

```
        printf(" %d\t\t%d\n", i + 1, allocation[i] + 1);
```

```
    return 0;
}
```

OUT PUT:

```
Process No.      Block No.
1                2
2                5
3                2
4                0

...Program finished with exit code 0
Press ENTER to exit console. 
```

24)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
```

```
int main() {
    int fd;
    char *text = "Hello, UNIX System Calls!\n";
    char buffer[50];
    fd = open("example.txt", O_CREAT | O_WRONLY, 0644);
    write(fd, text, 30);
    close(fd);
    fd = open("example.txt", O_RDONLY);
    read(fd, buffer, 30);
    printf("%s", buffer);
    close(fd);
}
```



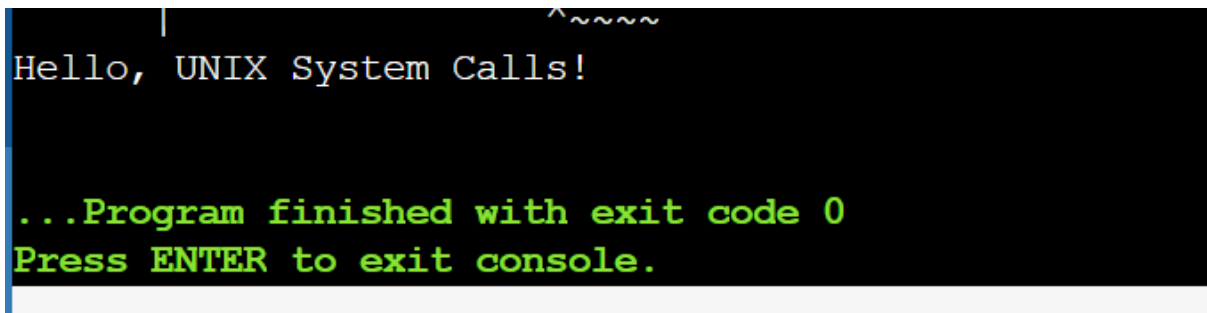
```

    unlink("example.txt");

    return 0;
}

```

**OUT PUT:**



A terminal window with a black background and a blue vertical bar on the left. The text is displayed in a monospaced font. The first line is "Hello, UNIX System Calls!" in white. The second line is "...Program finished with exit code 0" in green. The third line is "Press ENTER to exit console." in green. Above the first line, there is a small white cursor and some faint, illegible text.

```

Hello, UNIX System Calls!

...Program finished with exit code 0
Press ENTER to exit console.

```

25)

```

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/stat.h>

#include <dirent.h>

int main() {

    int fd = open("example.txt", O_RDWR | O_CREAT, 0644);

    fcntl(fd, F_SETFL, O_NONBLOCK);

    lseek(fd, 0, SEEK_SET);

    struct stat fileStat;

    stat("example.txt", &fileStat);

    printf("File size: %ld bytes\n", fileStat.st_size);

    DIR *dir = opendir(".");

    struct dirent *entry;

    while ((entry = readdir(dir)) != NULL) {

        printf("Found: %s\n", entry->d_name);

    }

    closedir(dir);

    close(fd);

    return 0;
}

```

```
}
```

OUT PUT:

```
File size: 0 bytes
Found: .
Found: ..
Found: main.c
Found: a.out
Found: example.txt

...Program finished with exit code 0
Press ENTER to exit console. 
```

26)

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

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

```
void createFile(const char *filename) {
```

```
    FILE *file = fopen(filename, "w");
```

```
    if (file) fclose(file);
```

```
}
```

```
void writeFile(const char *filename, const char *content) {
```

```
    FILE *file = fopen(filename, "a");
```

```
    if (file) {
```

```
        fputs(content, file);
```

```
        fclose(file);
```

```
    }
```

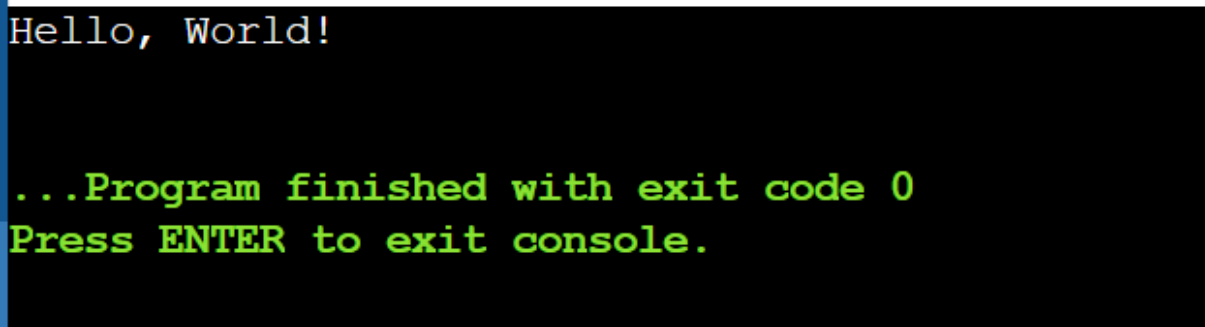
```
}
```

```
void readFile(const char *filename) {  
    char buffer[100];  
    FILE *file = fopen(filename, "r");  
    if (file) {  
        while (fgets(buffer, sizeof(buffer), file)) {  
            printf("%s", buffer);  
        }  
        fclose(file);  
    }  
}
```

```
void deleteFile(const char *filename) {  
    remove(filename);  
}
```

```
int main() {  
    const char *filename = "example.txt";  
    createFile(filename);  
    writeFile(filename, "Hello, World!\n");  
    readFile(filename);  
    deleteFile(filename);  
    return 0;  
}
```

**OUT PUT:**

A screenshot of a terminal window with a black background. The text "Hello, World!" is displayed in a white, monospaced font. Below it, there is a green message: "...Program finished with exit code 0" and "Press ENTER to exit console." in a green, monospaced font. The terminal window has a blue title bar at the top.

```
Hello, World!
```

```
...Program finished with exit code 0  
Press ENTER to exit console.
```

27)

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

```
#include <dirent.h>
```

```
int main() {
```

```
    struct dirent *entry;
```

```
    DIR *dp = opendir(".");
```

```
    if (dp == NULL) {
```

```
        perror("opendir");
```

```
        return 1;
```

```
    }
```

```
    while ((entry = readdir(dp)) != NULL) {
```

```
        printf("%s\n", entry->d_name);
```

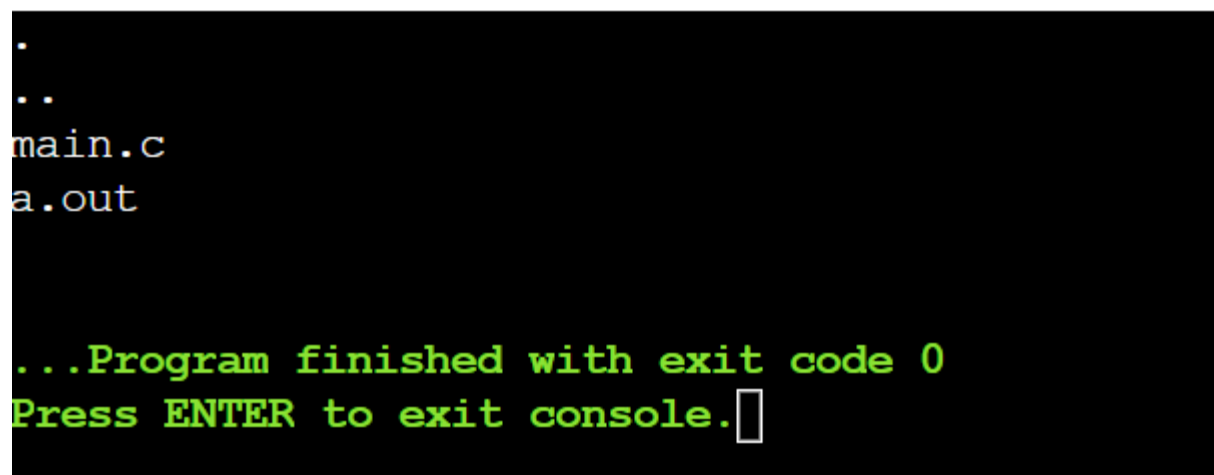
```
    }
```

```
    closedir(dp);
```

```
    return 0;
```

```
}
```

OUT PUT:

A terminal window with a black background and green text. It shows the output of a program that lists the contents of the current directory. The output is:   
.  
..  
main.c  
a.out  
  
...Program finished with exit code 0  
Press ENTER to exit console.  
The cursor is at the end of the last line.

```
.  
..  
main.c  
a.out  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

28)

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

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

```
int main(int argc, char *argv[]) {
```

```
    if (argc != 3) {
```

```
        printf("Usage: %s <pattern> <file>\n", argv[0]);
```

```
        return 1;
```

```
    }
```

```
    FILE *file = fopen(argv[2], "r");
```

```
    if (!file) {
```

```
        perror("File opening failed");
```

```
        return 1;
```

```
    }
```

```
    char line[256];
```

```
    while (fgets(line, sizeof(line), file)) {
```

```
        if (strstr(line, argv[1])) {
```

```
            printf("%s", line);
```

```
        }
```

```
    }
```

```
    fclose(file);
```

```
    return 0;
```

```
}
```

OUT PUT:

```
Usage: ./a.out <pattern> <file>
```

```
...Program finished with exit code 1  
Press ENTER to exit console.
```

29)

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

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

```
#include <pthread.h>
```

```
#include <semaphore.h>
```

```
#include <unistd.h>
```

```
#define NUM_THREADS 5
```

```
sem_t semaphore;
```

```
void* threadFunction(void* arg) {
```

```
    int id = *((int*)arg);
```

```
    sem_wait(&semaphore);
```

```
    printf("Thread %d is in the critical section.\n", id);
```

```
    sleep(1);
```

```
    printf("Thread %d is leaving the critical section.\n", id);
```

```
    sem_post(&semaphore);
```

```
    return NULL;
```

```
}
```

```
int main() {
```

```
    pthread_t threads[NUM_THREADS];
```

```
int thread_ids[NUM_THREADS];

sem_init(&semaphore, 0, 1);

for (int i = 0; i < NUM_THREADS; i++) {
    thread_ids[i] = i + 1;
    pthread_create(&threads[i], NULL, threadFunction, &thread_ids[i]);
}

for (int i = 0; i < NUM_THREADS; i++) {
    pthread_join(threads[i], NULL);
}

sem_destroy(&semaphore);
return 0;
}
```

#### OUT PUT:

```
Thread 1 is in the critical section.
Thread 1 is leaving the critical section.
Thread 4 is in the critical section.
Thread 4 is leaving the critical section.
Thread 2 is in the critical section.
Thread 2 is leaving the critical section.
Thread 3 is in the critical section.
Thread 3 is leaving the critical section.
Thread 5 is in the critical section.
Thread 5 is leaving the critical section.

...Program finished with exit code 0
Press ENTER to exit console.█
```

30)

1.

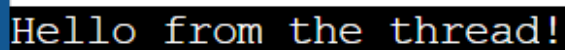
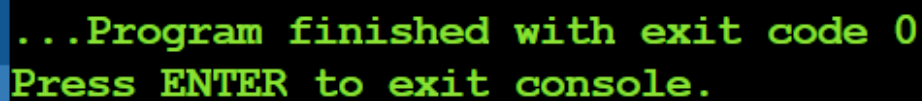
```
#include <stdio.h>

#include <pthread.h>

void* printMessage(void* msg) {
    printf("%s\n", (char*)msg);
    return NULL;
}

int main() {
    pthread_t thread;
    char* message = "Hello from the thread!";
    pthread_create(&thread, NULL, printMessage, (void*)message);
    pthread_join(thread, NULL);
    return 0;
}
```

OUT PUT:

A terminal window with a black background and a blue vertical bar on the left. The text "Hello from the thread!" is displayed in a light blue monospace font.The same terminal window as above, now showing two lines of green text: "...Program finished with exit code 0" and "Press ENTER to exit console." on separate lines.

2)

```
#include <stdio.h>

#include <pthread.h>

int counter = 0;

pthread_mutex_t lock;

void* incrementCounter(void* arg) {
    pthread_mutex_lock(&lock);
    counter++;
}
```



```

    pthread_mutex_unlock(&lock);
    return NULL;
}

int main() {
    pthread_t threads[5];
    pthread_mutex_init(&lock, NULL);
    for (int i = 0; i < 5; i++) pthread_create(&threads[i], NULL, incrementCounter, NULL);
    for (int i = 0; i < 5; i++) pthread_join(threads[i], NULL);
    printf("Counter: %d\n", counter);
    pthread_mutex_destroy(&lock);
    return 0;
}

```

OUT PUT:

```

Counter: 5

...Program finished with exit code 0
Press ENTER to exit console.

```

3)

```

#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

void* run(void* arg) {
    while (1) {
        printf("Thread running...\n");
        sleep(1);
    }
    return NULL;
}

```

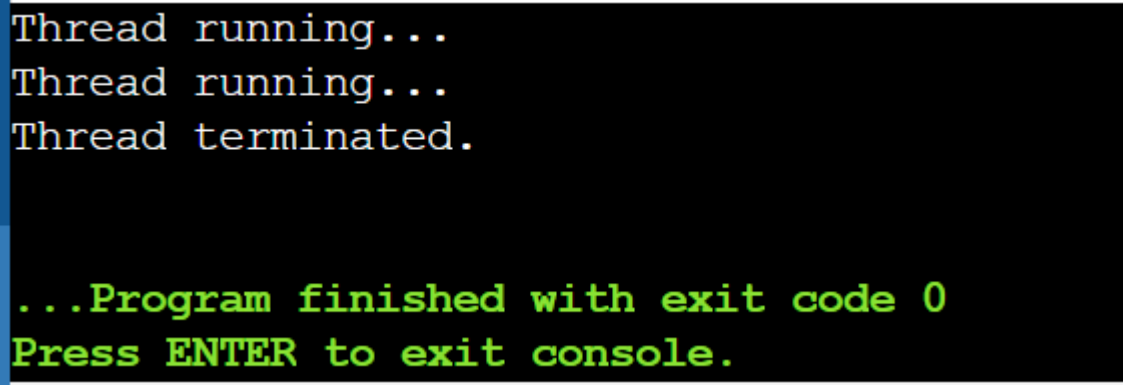
```

}

int main() {
    pthread_t thread;
    pthread_create(&thread, NULL, run, NULL);
    sleep(3);
    pthread_cancel(thread);
    pthread_join(thread, NULL);
    printf("Thread terminated.\n");
    return 0;
}

```

OUT PUT:



```

Thread running...
Thread running...
Thread terminated.

...Program finished with exit code 0
Press ENTER to exit console.

```

31)

```

#include <stdio.h>

#define FRAME_SIZE 3
#define PAGE_COUNT 5

int main() {
    int pages[PAGE_COUNT] = {0, 1, 2, 3, 4};
    int frames[FRAME_SIZE] = {-1, -1, -1};
    int pageFaults = 0, i, j, k;
    for (i = 0; i < PAGE_COUNT; i++) {
        int found = 0;
        for (j = 0; j < FRAME_SIZE; j++) {
            if (frames[j] == pages[i]) {

```

```

        found = 1; break;
    }
}
if (!found) {
    frames[pageFaults % FRAME_SIZE] = pages[i];
    pageFaults++;
}
printf("Page: %d | Frames: ", pages[i]);
for (k = 0; k < FRAME_SIZE; k++) printf("%d ", frames[k]);
printf("\n");
}
printf("Total Page Faults: %d\n", pageFaults);
return 0;
}

```

OUT PUT:

```

Page: 0 | Frames: 0 -1 -1
Page: 1 | Frames: 0 1 -1
Page: 2 | Frames: 0 1 2
Page: 3 | Frames: 3 1 2
Page: 4 | Frames: 3 4 2
Total Page Faults: 5

```

```

...Program finished with exit code 0
Press ENTER to exit console.

```

32)

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

```

int main() {

    int pages[10], frame[3], n, m, i, j, k, pos, flag, pageFaults = 0;

    printf("Enter number of pages: ");

    scanf("%d", &n);

    printf("Enter pages: ");

    for (i = 0; i < n; i++) scanf("%d", &pages[i]);


    for (i = 0; i < 3; i++) frame[i] = -1;


    for (i = 0; i < n; i++) {

        flag = 0;

        for (j = 0; j < 3; j++) if (frame[j] == pages[i]) flag = 1;

        if (!flag) {

            pageFaults++;

            pos = -1;

            for (j = 0; j < 3; j++) {

                if (frame[j] == -1) { pos = j; break; }

                for (k = i - 1; k >= 0; k--) if (frame[j] == pages[k]) { pos = j; break; }

            }

            if (pos == -1) pos = 0;

            frame[pos] = pages[i];

        }

    }

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

    return 0;

}

```

OUT PUT:

```
Enter number of pages: 4
Enter pages: 3 4 6 5
Page Faults: 4

...Program finished with exit code 0
Press ENTER to exit console.
```

33)

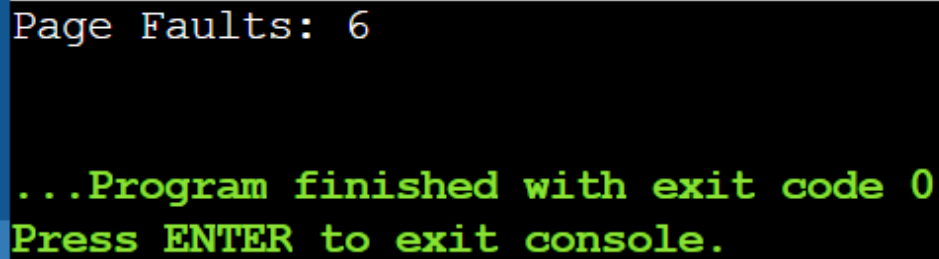
```
#include <stdio.h>
```

```
int main() {
    int pages[] = {0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};
    int frames[3] = {-1, -1, -1}, pageFaults = 0, n = 12, f = 3;

    for (int i = 0; i < n; i++) {
        int j, found = 0;
        for (j = 0; j < f; j++) if (frames[j] == pages[i]) found = 1;
        if (!found) {
            int replace = -1, farthest = -1;
            for (j = 0; j < f; j++) {
                int k;
                for (k = i + 1; k < n; k++) {
                    if (frames[j] == pages[k]) break;
                }
                if (k > farthest) { farthest = k; replace = j; }
            }
            frames[replace] = pages[i];
            pageFaults++;
        }
    }
}
```

```
printf("Page Faults: %d\n", pageFaults);  
return 0;  
}
```

OUT PUT:



```
Page Faults: 6  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

34)

```
#include <stdio.h>  
  
#define MAX_RECORDS 100  
  
void readRecords(char records[MAX_RECORDS][50], int count) {  
    for (int i = 0; i < count; i++) {  
        printf("Record %d: %s\n", i + 1, records[i]);  
    }  
}  
  
int main() {  
    char records[MAX_RECORDS][50] = {  
        "Record 1: Data A",  
        "Record 2: Data B",  
        "Record 3: Data C"  
    };  
  
    int count = 3;  
  
    readRecords(records, count);  
  
    return 0;  
}
```

OUT PUT:

```
Record 1: Record 1: Data A
Record 2: Record 2: Data B
Record 3: Record 3: Data C

...Program finished with exit code 0
Press ENTER to exit console.
```