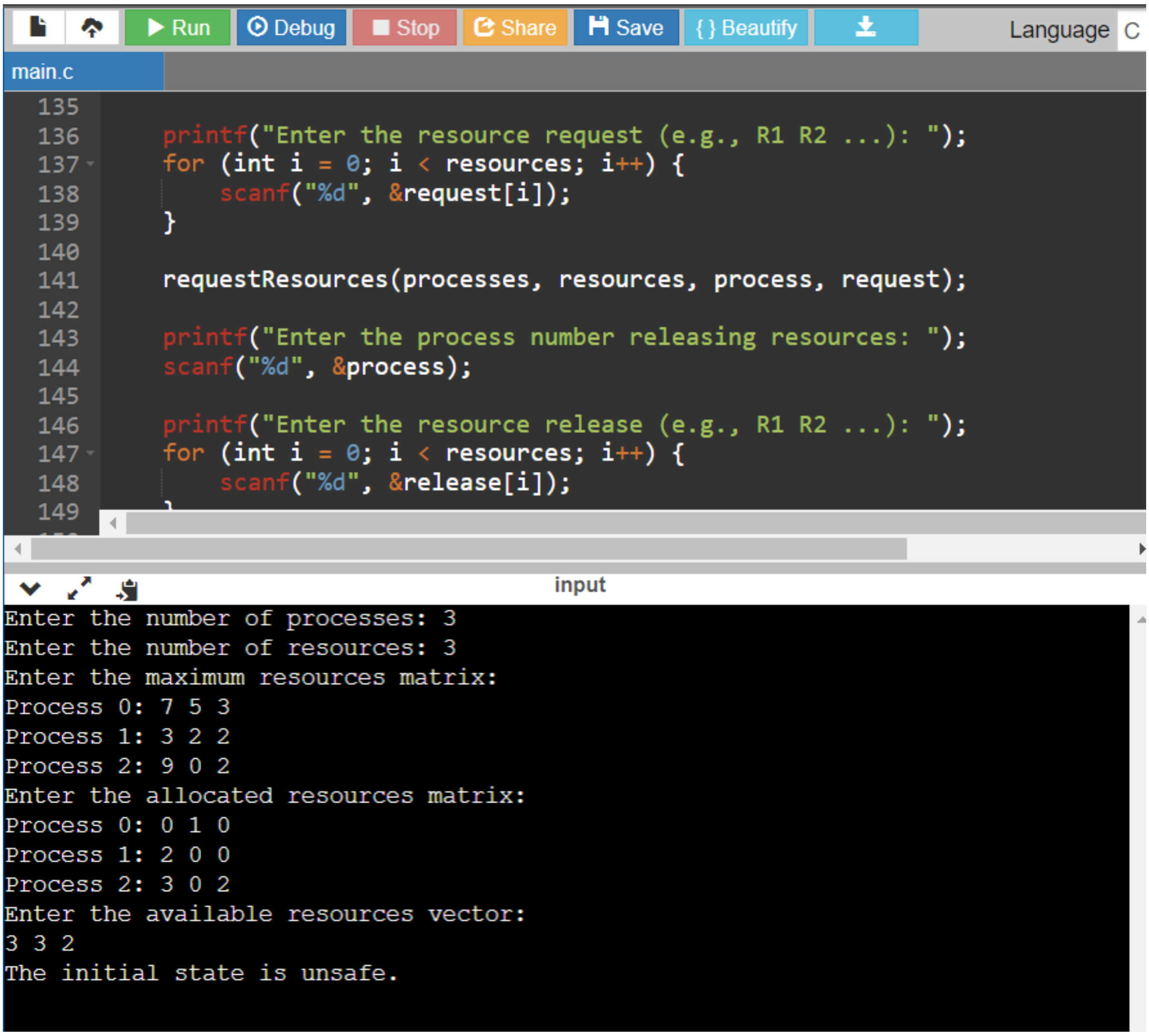
Q9. C program to implement Banker’s Algorithm

Source Code

```
#include <stdio.h>

#define MAX_PROCESSES 10
#define MAX_RESOURCES 10

int available[MAX_RESOURCES];
int max[MAX_PROCESSES][MAX_RESOURCES];
int allocated[MAX_PROCESSES][MAX_RESOURCES];
int need[MAX_PROCESSES][MAX_RESOURCES];

int isSafe(int processes, int resources) {
    int work[MAX_RESOURCES];
    int finish[MAX_PROCESSES] = {0};

    for (int i = 0; i < resources; i++) {
        work[i] = available[i];
    }

    int count = 0;
    while (count < processes) {
        int found = 0;
        for (int i = 0; i < processes; i++) {
            if (!finish[i]) {
                int j;
                for (j = 0; j < resources; j++) {
                    if (need[i][j] > work[j]) {
                        break;
                    }
                }

                if (j == resources) {
                    for (int k = 0; k < resources; k++) {
                        work[k] += allocated[i][k];
                    }
                    finish[i] = 1;
                    found = 1;
                    count++;
                }
            }
        }

        if (!found) {
            return 0;
        }
    }

    return 1;
}

void requestResources(int processes, int resources, int process, int request[]) {
    for (int i = 0; i < resources; i++) {
        if (request[i] > need[process][i] || request[i] > available[i]) {
            printf("Invalid request. The request exceeds the maximum need or available
resources.\n");
            return;
        }
    }

    for (int i = 0; i < resources; i++) {
        allocated[process][i] += request[i];
        available[i] -= request[i];
        need[process][i] -= request[i];
    }

    if (isSafe(processes, resources)) {
        printf("Request granted. The system is in a safe state.\n");
    } else {
        for (int i = 0; i < resources; i++) {
            allocated[process][i] -= request[i];
            available[i] += request[i];
            need[process][i] += request[i];
        }
        printf("Request denied. The system would be in an unsafe state.\n");
    }
}

void releaseResources(int resources, int process, int release[]) {
    for (int i = 0; i < resources; i++) {
        if (release[i] > allocated[process][i]) {
            printf("Invalid release. The release exceeds the allocated resources.\n");
            return;
        }
    }

    for (int i = 0; i < resources; i++) {
        allocated[process][i] -= release[i];
        available[i] += release[i];
    }
    printf("Resources released. The system is in a safe state.\n");
}

int main() {
    int processes, resources;

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

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

    printf("Enter the maximum resources matrix:\n");
    for (int i = 0; i < processes; i++) {
        printf("Process %d: ", i);
        for (int j = 0; j < resources; j++) {
            scanf("%d", &max[i][j]);
        }
    }

    printf("Enter the allocated resources matrix:\n");
    for (int i = 0; i < processes; i++) {
        printf("Process %d: ", i);
        for (int j = 0; j < resources; j++) {
            scanf("%d", &allocated[i][j]);
            need[i][j] = max[i][j] - allocated[i][j];
        }
    }

    printf("Enter the available resources vector:\n");
```

```
for (int i = 0; i < resources; i++) {
    scanf("%d", &available[i]);
}

if (isSafe(processes, resources)) {
    printf("The initial state is safe.\n");
} else {
    printf("The initial state is unsafe.\n");
    return 1;
}

// Demonstrate resource request and release
int process, request[MAX_RESOURCES], release[MAX_RESOURCES];

printf("Enter the process number requesting resources: ");
scanf("%d", &process);

printf("Enter the resource request (e.g., R1 R2 ...): ");
for (int i = 0; i < resources; i++) {
    scanf("%d", &request[i]);
}
requestResources(processes, resources, process, request);
printf("Enter the process number releasing resources: ");
scanf("%d", &process);
printf("Enter the resource release (e.g., R1 R2 ...): ");
for (int i = 0; i < resources; i++) {
    scanf("%d", &release[i]);
}
releaseResources(resources, process, release);
return 0;}
```

Output



```
    printf("Enter the resource request (e.g., R1 R2 ...): ");
    for (int i = 0; i < resources; i++) {
        scanf("%d", &request[i]);
    }

    requestResources(processes, resources, process, request);

    printf("Enter the process number releasing resources: ");
    scanf("%d", &process);

    printf("Enter the resource release (e.g., R1 R2 ...): ");
    for (int i = 0; i < resources; i++) {
        scanf("%d", &release[i]);
    }
```

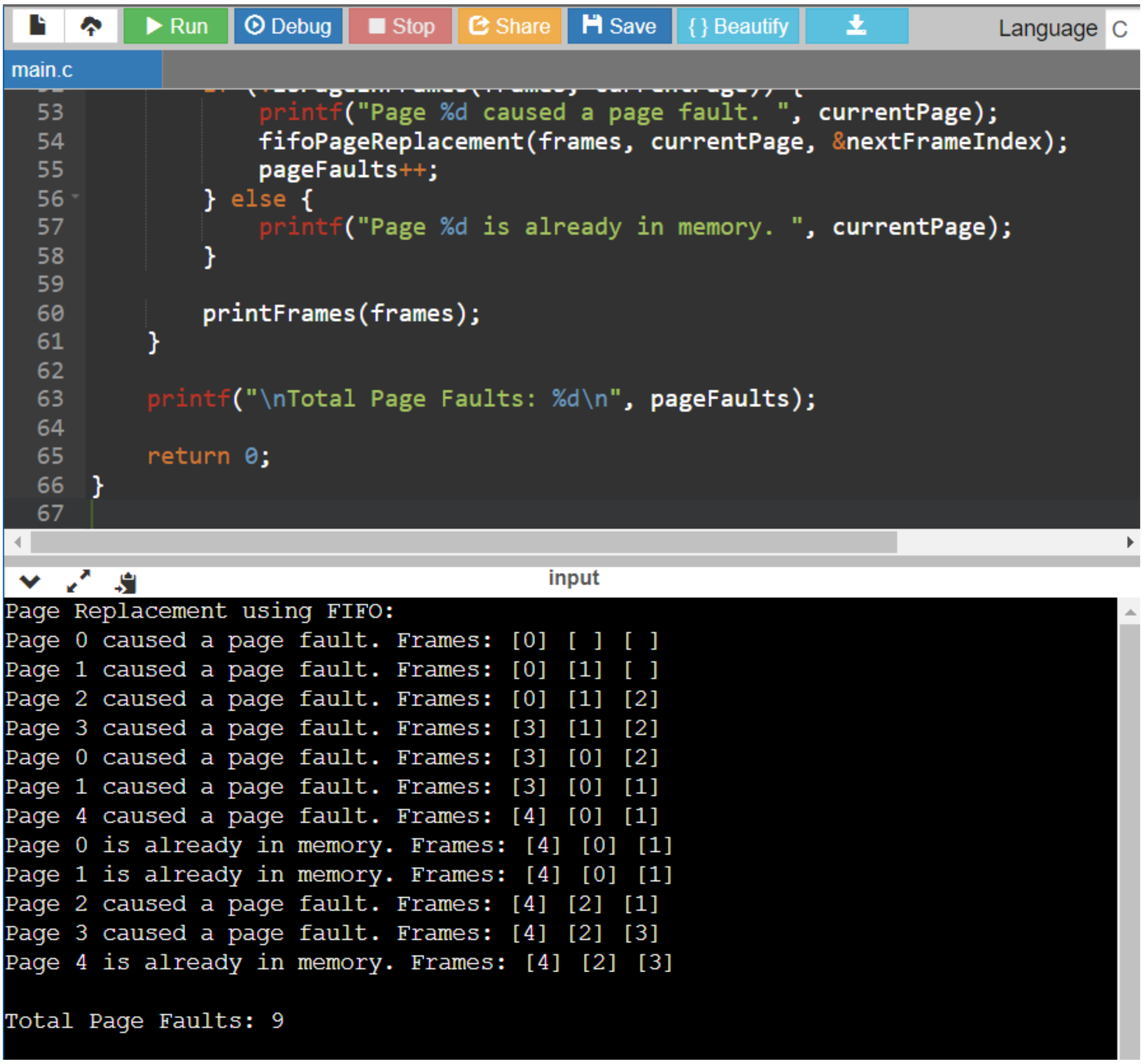
```
Enter the number of processes: 3
Enter the number of resources: 3
Enter the maximum resources matrix:
Process 0: 7 5 3
Process 1: 3 2 2
Process 2: 9 0 2
Enter the allocated resources matrix:
Process 0: 0 1 0
Process 1: 2 0 0
Process 2: 3 0 2
Enter the available resources vector:
3 3 2
The initial state is unsafe.
```

Q10. C program to implement first in first out page replacement policy

Source Code

```
#include <stdio.h>

#define MAX_FRAMES 3

void initializeFrames(int frames[MAX_FRAMES]) {
    for (int i = 0; i < MAX_FRAMES; i++) {
        frames[i] = -1;  // -1 indicates an empty frame
    }
}

void printFrames(int frames[MAX_FRAMES]) {
    printf("Frames: ");
    for (int i = 0; i < MAX_FRAMES; i++) {
        if (frames[i] == -1) {
            printf("[ ] ");
        } else {
            printf("[%d] ", frames[i]);
        }
    }
    printf("\n");
}

int isPageInFrames(int frames[MAX_FRAMES], int page) {
    for (int i = 0; i < MAX_FRAMES; i++) {
        if (frames[i] == page) {
            return 1;  // Page is already in frames
        }
    }
    return 0;  // Page is not in frames
}
void fifoPageReplacement(int frames[MAX_FRAMES], int page, int *nextFrameIndex) {
    frames[*nextFrameIndex] = page;
    *nextFrameIndex = (*nextFrameIndex + 1) % MAX_FRAMES;}
int main() {
    int frames[MAX_FRAMES];
    initializeFrames(frames);
    int pageSequence[] = {0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4};
    int pageSequenceSize = sizeof(pageSequence) / sizeof(pageSequence[0]);
    int pageFaults = 0;
    int nextFrameIndex = 0;
    printf("Page Replacement using FIFO:\n");
    for (int i = 0; i < pageSequenceSize; i++) {
        int currentPage = pageSequence[i];
        if (!isPageInFrames(frames, currentPage)) {
            printf("Page %d caused a page fault. ", currentPage);
            fifoPageReplacement(frames, currentPage, &nextFrameIndex);
            pageFaults++;
        } else {
            printf("Page %d is already in memory. ", currentPage);}
        printFrames(frames);}
    printf("\nTotal Page Faults: %d\n", pageFaults);
    return 0;
}
```

Output



```c
        if (!isPageInFrames(frames, currentPage)) {
            printf("Page %d caused a page fault. ", currentPage);
            fifoPageReplacement(frames, currentPage, &nextFrameIndex);
            pageFaults++;
        } else {
            printf("Page %d is already in memory. ", currentPage);
        }

        printFrames(frames);
    }

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

    return 0;
}
```

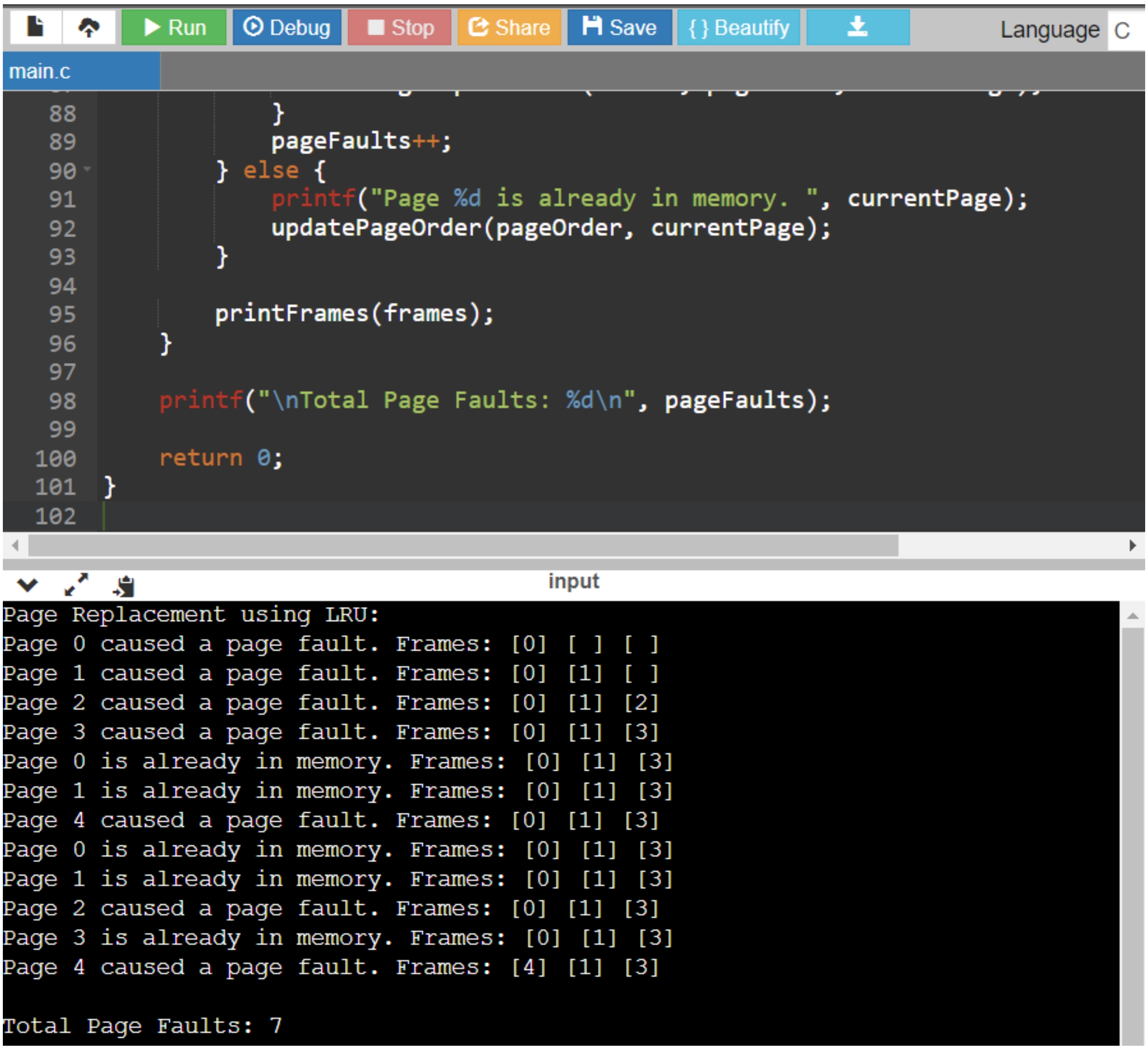
```
Page Replacement using FIFO:
Page 0 caused a page fault. Frames: [0] [ ] [ ]
Page 1 caused a page fault. Frames: [0] [1] [ ]
Page 2 caused a page fault. Frames: [0] [1] [2]
Page 3 caused a page fault. Frames: [3] [1] [2]
Page 0 caused a page fault. Frames: [3] [0] [2]
Page 1 caused a page fault. Frames: [3] [0] [1]
Page 4 caused a page fault. Frames: [4] [0] [1]
Page 0 is already in memory. Frames: [4] [0] [1]
Page 1 is already in memory. Frames: [4] [0] [1]
Page 2 caused a page fault. Frames: [4] [2] [1]
Page 3 caused a page fault. Frames: [4] [2] [3]
Page 4 is already in memory. Frames: [4] [2] [3]

Total Page Faults: 9
```

Q11. C program to implement least recently used page replacement policy

Source Code

```
#include <stdio.h>

#define MAX_FRAMES 3

void initializeFrames(int frames[MAX_FRAMES]) {
    for (int i = 0; i < MAX_FRAMES; i++) {
        frames[i] = -1;  // -1 indicates an empty frame
    }
}

void printFrames(int frames[MAX_FRAMES]) {
    printf("Frames: ");
    for (int i = 0; i < MAX_FRAMES; i++) {
        if (frames[i] == -1) {
            printf("[ ] ");
        } else {
            printf("[%d] ", frames[i]);
        }
    }
    printf("\n");
}

int isPageInFrames(int frames[MAX_FRAMES], int page) {
    for (int i = 0; i < MAX_FRAMES; i++) {
        if (frames[i] == page) {
            return 1;  // Page is already in frames
        }
    }
    return 0;  // Page is not in frames
}

int getLRUPage(int pageOrder[MAX_FRAMES]) {
    return pageOrder[MAX_FRAMES - 1];
}

void updatePageOrder(int pageOrder[MAX_FRAMES], int currentPage) {
    // Move the current page to the front of the page order
    for (int i = 0; i < MAX_FRAMES; i++) {
        if (pageOrder[i] == currentPage) {
            for (int j = i; j > 0; j--) {
                pageOrder[j] = pageOrder[j - 1];
            }
            pageOrder[0] = currentPage;
            break;
        }
    }
}

void lruPageReplacement(int frames[MAX_FRAMES], int pageOrder[MAX_FRAMES], int
page) {
    int leastRecentlyUsedPage = getLRUPage(pageOrder);

    for (int i = 0; i < MAX_FRAMES; i++) {
        if (frames[i] == leastRecentlyUsedPage) {
            frames[i] = page;
            break;
        }
    }
    updatePageOrder(pageOrder, page);
}

int main() {
    int frames[MAX_FRAMES];
    initializeFrames(frames);

    int pageSequence[] = {0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4};
    int pageSequenceSize = sizeof(pageSequence) / sizeof(pageSequence[0]);

    int pageOrder[MAX_FRAMES];
    for (int i = 0; i < MAX_FRAMES; i++) {
        pageOrder[i] = -1;  // Initialize page order
    }

    int pageFaults = 0;

    printf("Page Replacement using LRU:\n");

    for (int i = 0; i < pageSequenceSize; i++) {
        int currentPage = pageSequence[i];

        if (!isPageInFrames(frames, currentPage)) {
            printf("Page %d caused a page fault. ", currentPage);
            if (pageFaults < MAX_FRAMES) {
                frames[pageFaults] = currentPage;
                pageOrder[pageFaults] = currentPage;
            } else {
                lruPageReplacement(frames, pageOrder, currentPage);
            }
            pageFaults++;
        } else {
            printf("Page %d is already in memory. ", currentPage);
            updatePageOrder(pageOrder, currentPage);
        }

        printFrames(frames);
    }

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

    return 0;
}
```

Output



```c
            }
            pageFaults++;
        } else {
            printf("Page %d is already in memory. ", currentPage);
            updatePageOrder(pageOrder, currentPage);
        }

        printFrames(frames);
    }

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

    return 0;
}
```

```
Page Replacement using LRU:
Page 0 caused a page fault. Frames: [0] [ ] [ ]
Page 1 caused a page fault. Frames: [0] [1] [ ]
Page 2 caused a page fault. Frames: [0] [1] [2]
Page 3 caused a page fault. Frames: [0] [1] [3]
Page 0 is already in memory. Frames: [0] [1] [3]
Page 1 is already in memory. Frames: [0] [1] [3]
Page 4 caused a page fault. Frames: [0] [1] [3]
Page 0 is already in memory. Frames: [0] [1] [3]
Page 1 is already in memory. Frames: [0] [1] [3]
Page 2 caused a page fault. Frames: [0] [1] [3]
Page 3 is already in memory. Frames: [0] [1] [3]
Page 4 caused a page fault. Frames: [4] [1] [3]

Total Page Faults: 7
```

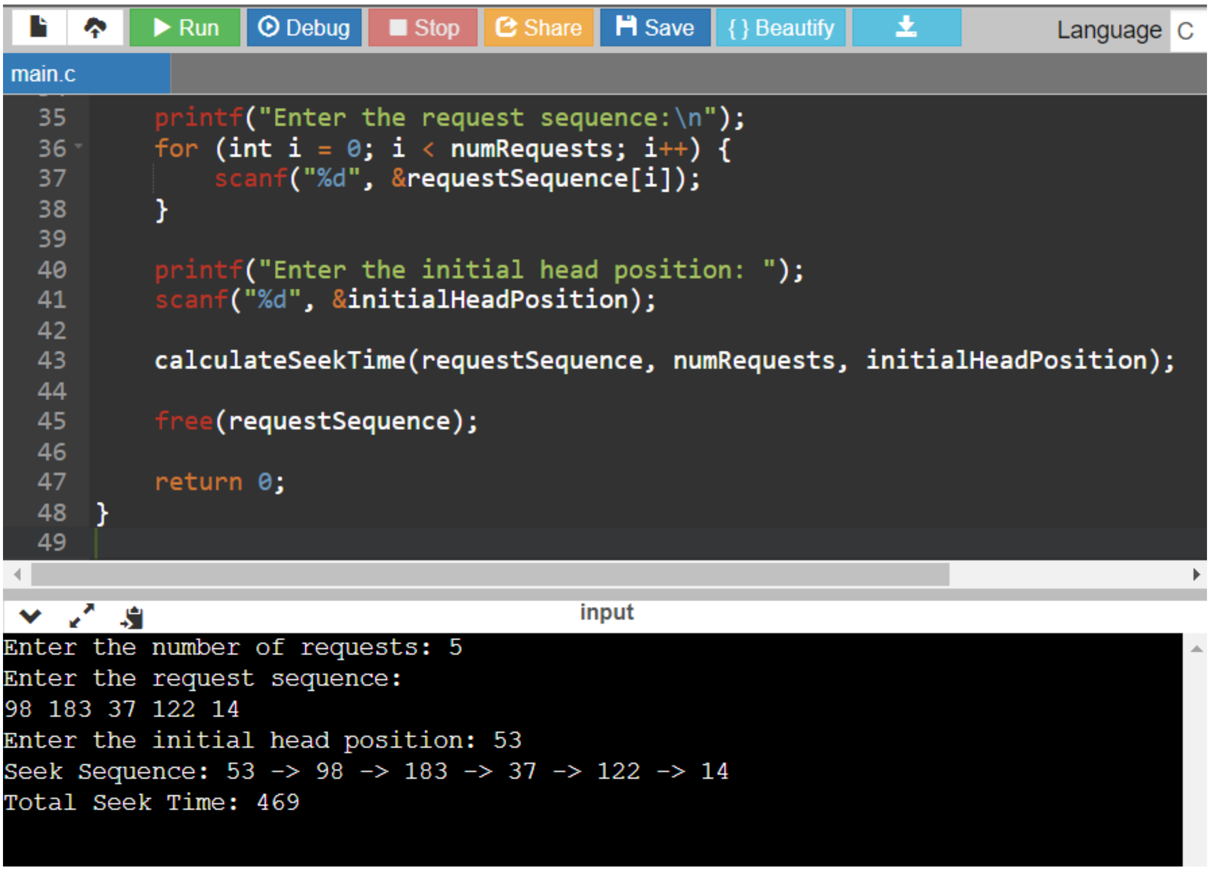
\

Q12. C program to implement FCFS Disk Scheduling Algorithm

Source Code

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

void calculateSeekTime(int requestSequence[], int numRequests, int initialHeadPosition) {
    int seekTime = 0;
    int currentHeadPosition = initialHeadPosition;

    printf("Seek Sequence: %d", currentHeadPosition);

    for (int i = 0; i < numRequests; i++) {
        int distance = abs(requestSequence[i] - currentHeadPosition);
        seekTime += distance;
        currentHeadPosition = requestSequence[i];

        printf(" -> %d", currentHeadPosition);
    }

    printf("\nTotal Seek Time: %d\n", seekTime);
}

int main() {
    int numRequests;
    int initialHeadPosition;

    printf("Enter the number of requests: ");
    scanf("%d", &numRequests);

    int *requestSequence = (int *)malloc(numRequests * sizeof(int));
    if (requestSequence == NULL) {
        fprintf(stderr, "Memory allocation failed.\n");
        return 1; // Exit with an error code
    }

    printf("Enter the request sequence:\n");
    for (int i = 0; i < numRequests; i++) {
        scanf("%d", &requestSequence[i]);
    }

    printf("Enter the initial head position: ");
    scanf("%d", &initialHeadPosition);

    calculateSeekTime(requestSequence, numRequests, initialHeadPosition);

    free(requestSequence);

    return 0;
}
```

Output



```
    printf("Enter the request sequence:\n");
    for (int i = 0; i < numRequests; i++) {
        scanf("%d", &requestSequence[i]);
    }

    printf("Enter the initial head position: ");
    scanf("%d", &initialHeadPosition);

    calculateSeekTime(requestSequence, numRequests, initialHeadPosition);

    free(requestSequence);

    return 0;
}
```

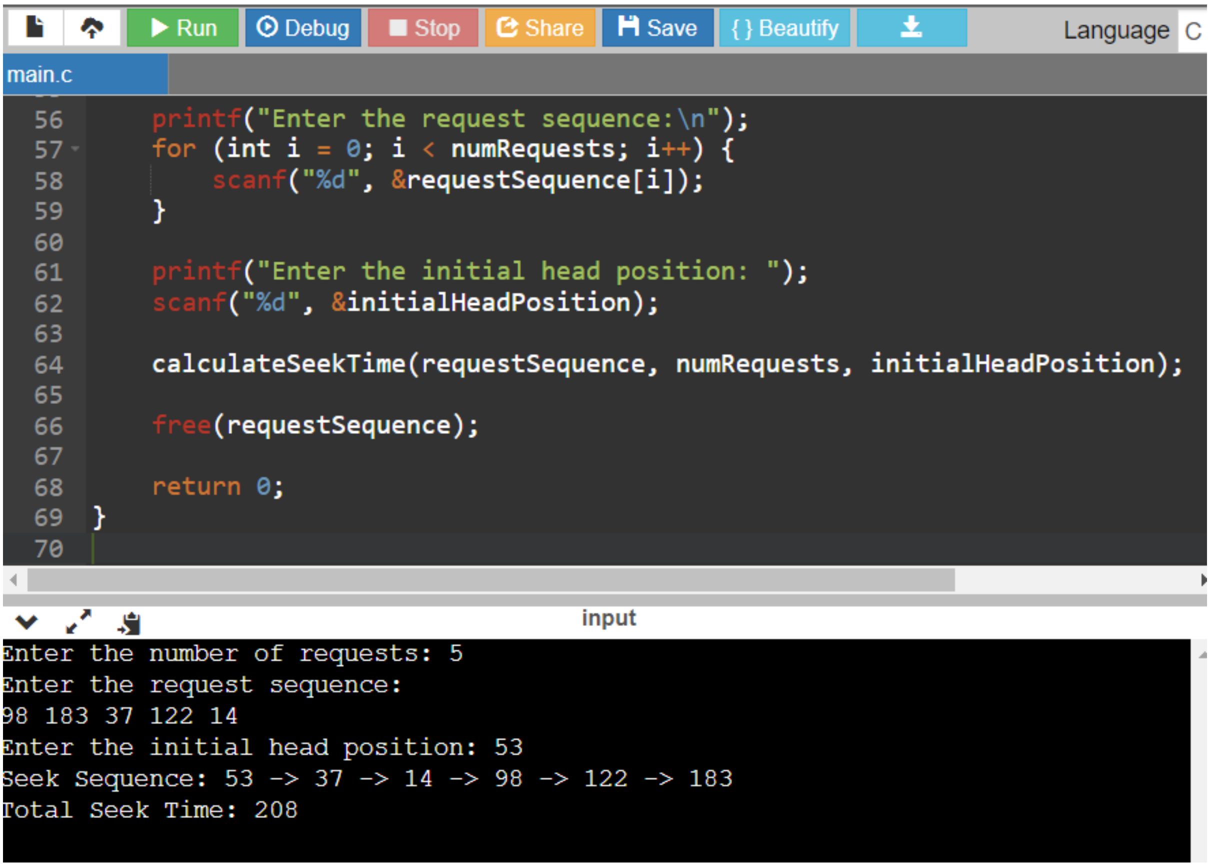
```
Enter the number of requests: 5
Enter the request sequence:
98 183 37 122 14
Enter the initial head position: 53
Seek Sequence: 53 -> 98 -> 183 -> 37 -> 122 -> 14
Total Seek Time: 469
```

Q13. C program to implement the SSTF Disk Scheduling Algorithm

Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h> // Include the header file for INT_MAX

// Function to calculate seek time using SSTF algorithm
void calculateSeekTime(int requestSequence[], int numRequests, int initialHeadPosition) {
    int seekTime = 0;
    int currentHeadPosition = initialHeadPosition;
    int visited[numRequests];

    for (int i = 0; i < numRequests; i++) {
        visited[i] = 0;  // Initialize all requests as not visited
    }

    printf("Seek Sequence: %d", currentHeadPosition);

    for (int i = 0; i < numRequests; i++) {
        int minDistance = INT_MAX; // Use INT_MAX from <limits.h>
        int nextRequest = -1;

        // Find the request with the shortest seek time
        for (int j = 0; j < numRequests; j++) {
            if (!visited[j]) {
                int distance = abs(requestSequence[j] - currentHeadPosition);
                if (distance < minDistance) {
                    minDistance = distance;
                    nextRequest = j;
                }
            }
        }

        visited[nextRequest] = 1;  // Mark the request as visited
        seekTime += minDistance;
        currentHeadPosition = requestSequence[nextRequest];

        printf(" -> %d", currentHeadPosition);
    }

    printf("\nTotal Seek Time: %d\n", seekTime);
}

int main() {
    int numRequests;
    int initialHeadPosition;

    printf("Enter the number of requests: ");
    scanf("%d", &numRequests);

    int *requestSequence = (int *)malloc(numRequests * sizeof(int));

    if (requestSequence == NULL) {
        fprintf(stderr, "Memory allocation failed.\n");
        return 1; // Exit with an error code
    }

    printf("Enter the request sequence:\n");
    for (int i = 0; i < numRequests; i++) {
        scanf("%d", &requestSequence[i]);
    }

    printf("Enter the initial head position: ");
    scanf("%d", &initialHeadPosition);

    calculateSeekTime(requestSequence, numRequests, initialHeadPosition);

    free(requestSequence);

    return 0;
}
```

Output



```
    printf("Enter the request sequence:\n");
    for (int i = 0; i < numRequests; i++) {
        scanf("%d", &requestSequence[i]);
    }

    printf("Enter the initial head position: ");
    scanf("%d", &initialHeadPosition);

    calculateSeekTime(requestSequence, numRequests, initialHeadPosition);

    free(requestSequence);

    return 0;
}
```

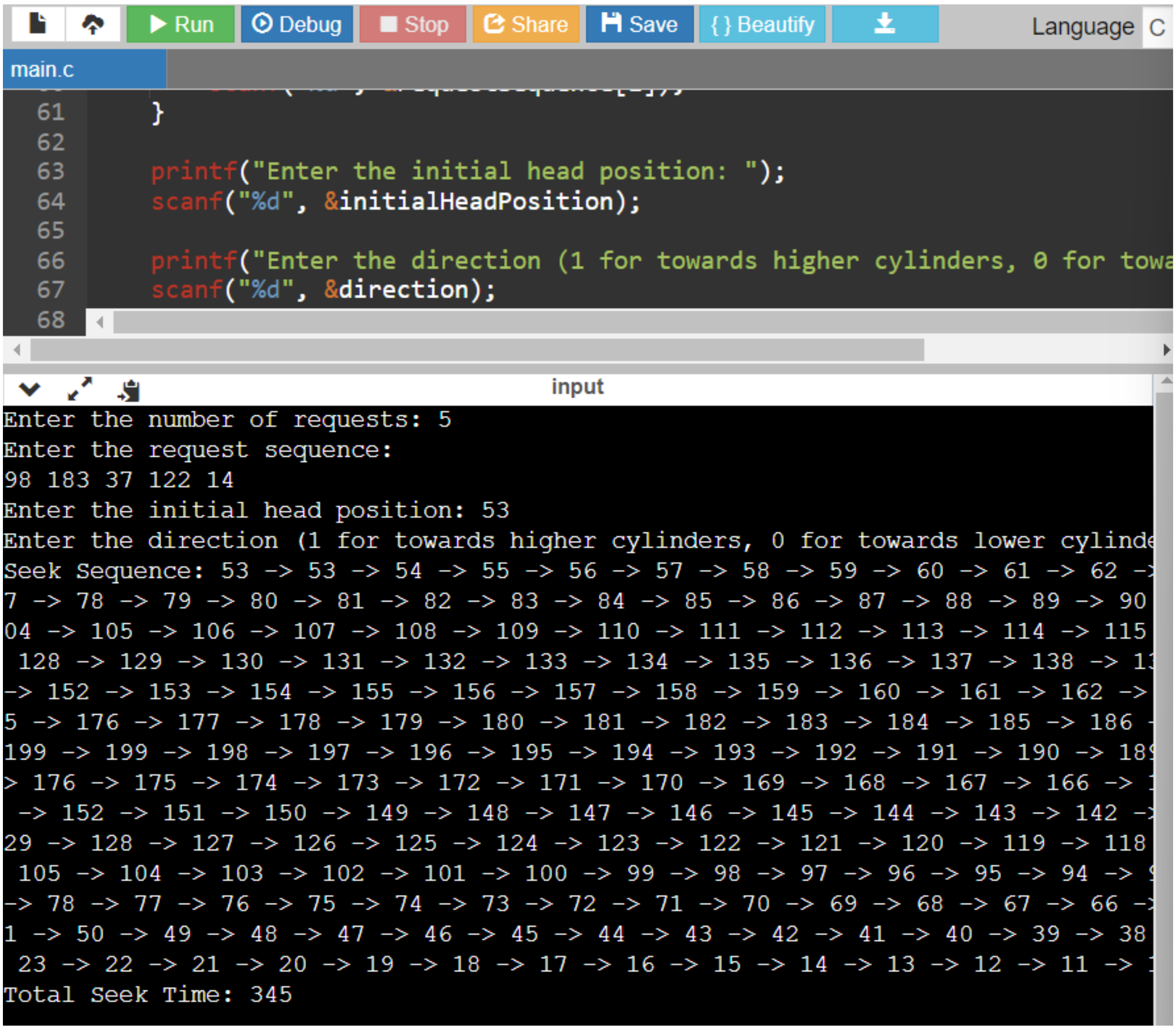
```
Enter the number of requests: 5
Enter the request sequence:
98 183 37 122 14
Enter the initial head position: 53
Seek Sequence: 53 -> 37 -> 14 -> 98 -> 122 -> 183
Total Seek Time: 208
```

Q14. C program to implement SCAN Disk Scheduling Algorithm

Source Code

```
#include <stdio.h>

#include <stdlib.h>


void calculateSeekTime(int requestSequence[], int numRequests, int initialHeadPosition, int
direction) {

    int seekTime = 0;

    int currentHeadPosition = initialHeadPosition;


    printf("Seek Sequence: %d", currentHeadPosition);


    if (direction == 1) {  // Move towards higher cylinder numbers

        // Go to the end of the disk

        for (int i = currentHeadPosition; i <= 199; i++) {

            printf(" -> %d", i);

            seekTime += abs(i - currentHeadPosition);

            currentHeadPosition = i;

        }


        // Go to the beginning of the disk

        for (int i = 199; i >= 0; i--) {

            printf(" -> %d", i);

            seekTime += abs(i - currentHeadPosition);

            currentHeadPosition = i;

        }

    } else {  // Move towards lower cylinder numbers

        // Go to the beginning of the disk

        for (int i = currentHeadPosition; i >= 0; i--) {

            printf(" -> %d", i);

            seekTime += abs(i - currentHeadPosition);
            currentHeadPosition = i;
        }

        // Go to the end of the disk
        for (int i = 0; i <= 199; i++) {
            printf(" -> %d", i);
            seekTime += abs(i - currentHeadPosition);
            currentHeadPosition = i;
        }
    }

    printf("\nTotal Seek Time: %d\n", seekTime);
}

int main() {
    int numRequests;
    int initialHeadPosition;
    int direction;

    printf("Enter the number of requests: ");
    scanf("%d", &numRequests);

    int *requestSequence = (int *)malloc(numRequests * sizeof(int));

    if (requestSequence == NULL) {
        fprintf(stderr, "Memory allocation failed.\n");
        return 1; // Exit with an error code
    }

    printf("Enter the request sequence:\n");
    for (int i = 0; i < numRequests; i++) {
        scanf("%d", &requestSequence[i]);
    }

    printf("Enter the initial head position: ");
    scanf("%d", &initialHeadPosition);

    printf("Enter the direction (1 for towards higher cylinders, 0 for towards lower cylinders): ");
    scanf("%d", &direction);

    calculateSeekTime(requestSequence, numRequests, initialHeadPosition, direction);

    free(requestSequence);

    return 0;
}
```

Output



```
    }

    printf("Enter the initial head position: ");
    scanf("%d", &initialHeadPosition);

    printf("Enter the direction (1 for towards higher cylinders, 0 for towa
    scanf("%d", &direction);

```

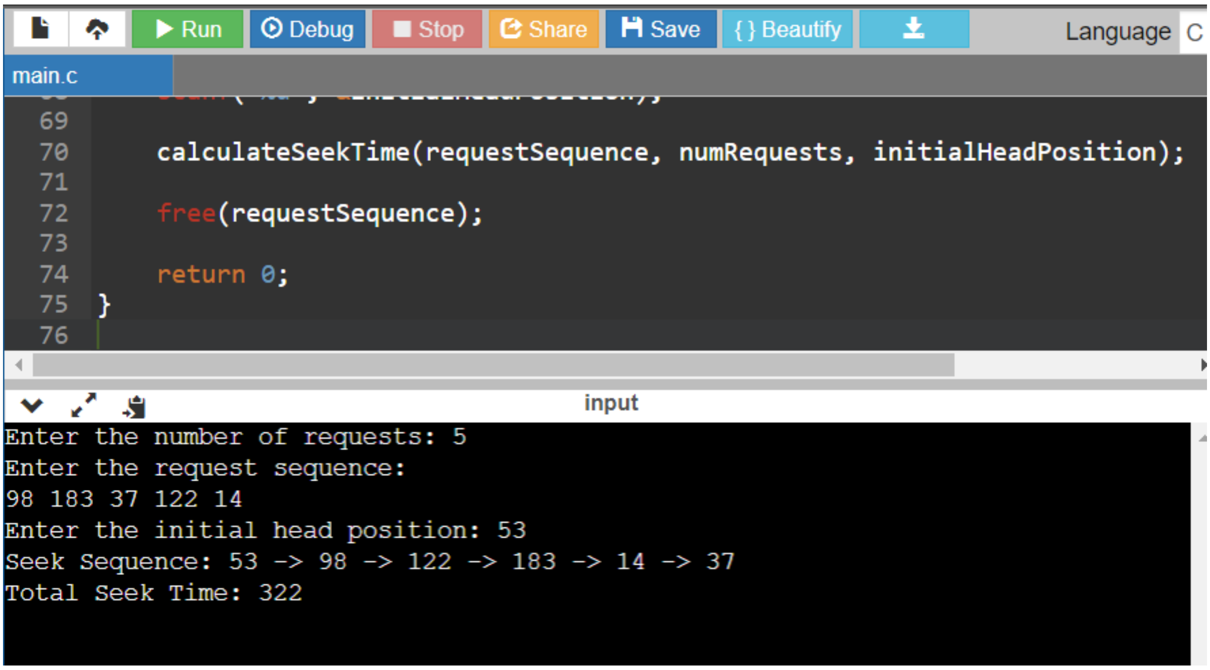
```
Enter the number of requests: 5
Enter the request sequence:
98 183 37 122 14
Enter the initial head position: 53
Enter the direction (1 for towards higher cylinders, 0 for towards lower cylinde
Seek Sequence: 53 -> 53 -> 54 -> 55 -> 56 -> 57 -> 58 -> 59 -> 60 -> 61 -> 62 ->
7 -> 78 -> 79 -> 80 -> 81 -> 82 -> 83 -> 84 -> 85 -> 86 -> 87 -> 88 -> 89 -> 90
04 -> 105 -> 106 -> 107 -> 108 -> 109 -> 110 -> 111 -> 112 -> 113 -> 114 -> 115
 128 -> 129 -> 130 -> 131 -> 132 -> 133 -> 134 -> 135 -> 136 -> 137 -> 138 -> 1
-> 152 -> 153 -> 154 -> 155 -> 156 -> 157 -> 158 -> 159 -> 160 -> 161 -> 162 ->
5 -> 176 -> 177 -> 178 -> 179 -> 180 -> 181 -> 182 -> 183 -> 184 -> 185 -> 186 -
199 -> 199 -> 198 -> 197 -> 196 -> 195 -> 194 -> 193 -> 192 -> 191 -> 190 -> 18
> 176 -> 175 -> 174 -> 173 -> 172 -> 171 -> 170 -> 169 -> 168 -> 167 -> 166 ->
 -> 152 -> 151 -> 150 -> 149 -> 148 -> 147 -> 146 -> 145 -> 144 -> 143 -> 142 ->
29 -> 128 -> 127 -> 126 -> 125 -> 124 -> 123 -> 122 -> 121 -> 120 -> 119 -> 118
 105 -> 104 -> 103 -> 102 -> 101 -> 100 -> 99 -> 98 -> 97 -> 96 -> 95 -> 94 ->
-> 78 -> 77 -> 76 -> 75 -> 74 -> 73 -> 72 -> 71 -> 70 -> 69 -> 68 -> 67 -> 66 ->
1 -> 50 -> 49 -> 48 -> 47 -> 46 -> 45 -> 44 -> 43 -> 42 -> 41 -> 40 -> 39 -> 38
 23 -> 22 -> 21 -> 20 -> 19 -> 18 -> 17 -> 16 -> 15 -> 14 -> 13 -> 12 -> 11 ->
Total Seek Time: 345
```

Q15. C program to implement C-SCAN Disk scheduling

Source Code

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

void calculateSeekTime(int requestSequence[], int numRequests, int initialHeadPosition) {
    int seekTime = 0;
    int currentHeadPosition = initialHeadPosition;

    printf("Seek Sequence: %d", currentHeadPosition);

    // Sort the request sequence
    for (int i = 0; i < numRequests - 1; i++) {
        for (int j = 0; j < numRequests - i - 1; j++) {
            if (requestSequence[j] > requestSequence[j + 1]) {
                // Swap the requests if they are out of order
                int temp = requestSequence[j];
                requestSequence[j] = requestSequence[j + 1];
                requestSequence[j + 1] = temp;
            }
        }
    }

    // Find the index where the current head position is located in the sorted sequence
    int index = 0;
    for (int i = 0; i < numRequests; i++) {
        if (requestSequence[i] >= currentHeadPosition) {
            index = i;
            break;
        }
    }

    // Go to the end of the disk
    for (int i = index; i < numRequests; i++) {
        printf(" -> %d", requestSequence[i]);
        seekTime += abs(requestSequence[i] - currentHeadPosition);
        currentHeadPosition = requestSequence[i];
    }

    // Go to the beginning of the disk
    for (int i = 0; i < index; i++) {
        printf(" -> %d", requestSequence[i]);
        seekTime += abs(requestSequence[i] - currentHeadPosition);
        currentHeadPosition = requestSequence[i];
    }

    printf("\nTotal Seek Time: %d\n", seekTime);
}

int main() {
    int numRequests;
    int initialHeadPosition;

    printf("Enter the number of requests: ");
    scanf("%d", &numRequests);

    int *requestSequence = (int *)malloc(numRequests * sizeof(int));

    if (requestSequence == NULL) {
        fprintf(stderr, "Memory allocation failed.\n");
        return 1; // Exit with an error code
    }

    printf("Enter the request sequence:\n");
    for (int i = 0; i < numRequests; i++) {
        scanf("%d", &requestSequence[i]);
    }

    printf("Enter the initial head position: ");
    scanf("%d", &initialHeadPosition);

    calculateSeekTime(requestSequence, numRequests, initialHeadPosition);

    free(requestSequence);

    return 0;
}
```

Output



```c
    calculateSeekTime(requestSequence, numRequests, initialHeadPosition);

    free(requestSequence);

    return 0;
}

```

```
Enter the number of requests: 5
Enter the request sequence:
98 183 37 122 14
Enter the initial head position: 53
Seek Sequence: 53 -> 98 -> 122 -> 183 -> 14 -> 37
Total Seek Time: 322
```