

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\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]);
}

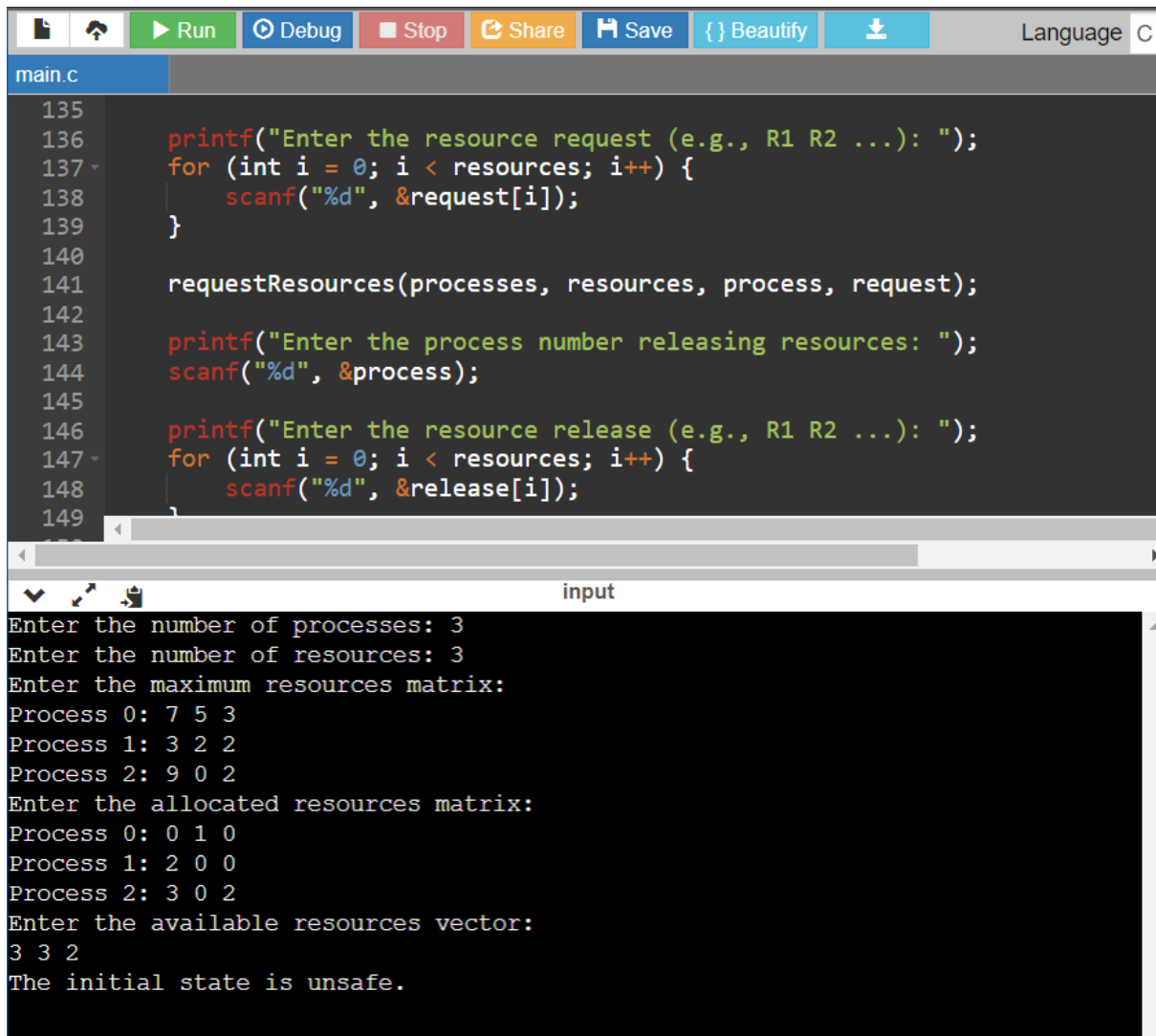
requestResources(processes, resources, process, request);
printf("Enter the process number releasing resources: ");
scanf("%d", &process);

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



The image shows a C code editor window with a toolbar at the top containing icons for file operations, a 'Run' button, 'Debug', 'Stop', 'Share', 'Save', 'Beautify', and a download icon. The language is set to 'C'. The code in 'main.c' includes a loop for resource requests, a function call 'requestResources', a loop for process releases, and a final printf statement. Below the code is a terminal window titled 'input' showing the program's execution with user inputs and the resulting output.

```
135
136     printf("Enter the resource request (e.g., R1 R2 ...): ");
137     for (int i = 0; i < resources; i++) {
138         scanf("%d", &request[i]);
139     }
140
141     requestResources(processes, resources, process, request);
142
143     printf("Enter the process number releasing resources: ");
144     scanf("%d", &process);
145
146     printf("Enter the resource release (e.g., R1 R2 ...): ");
147     for (int i = 0; i < resources; i++) {
148         scanf("%d", &release[i]);
149     }
150
151     printf("The initial state is unsafe.");
```

input

```
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
        }
    }
}
```

```

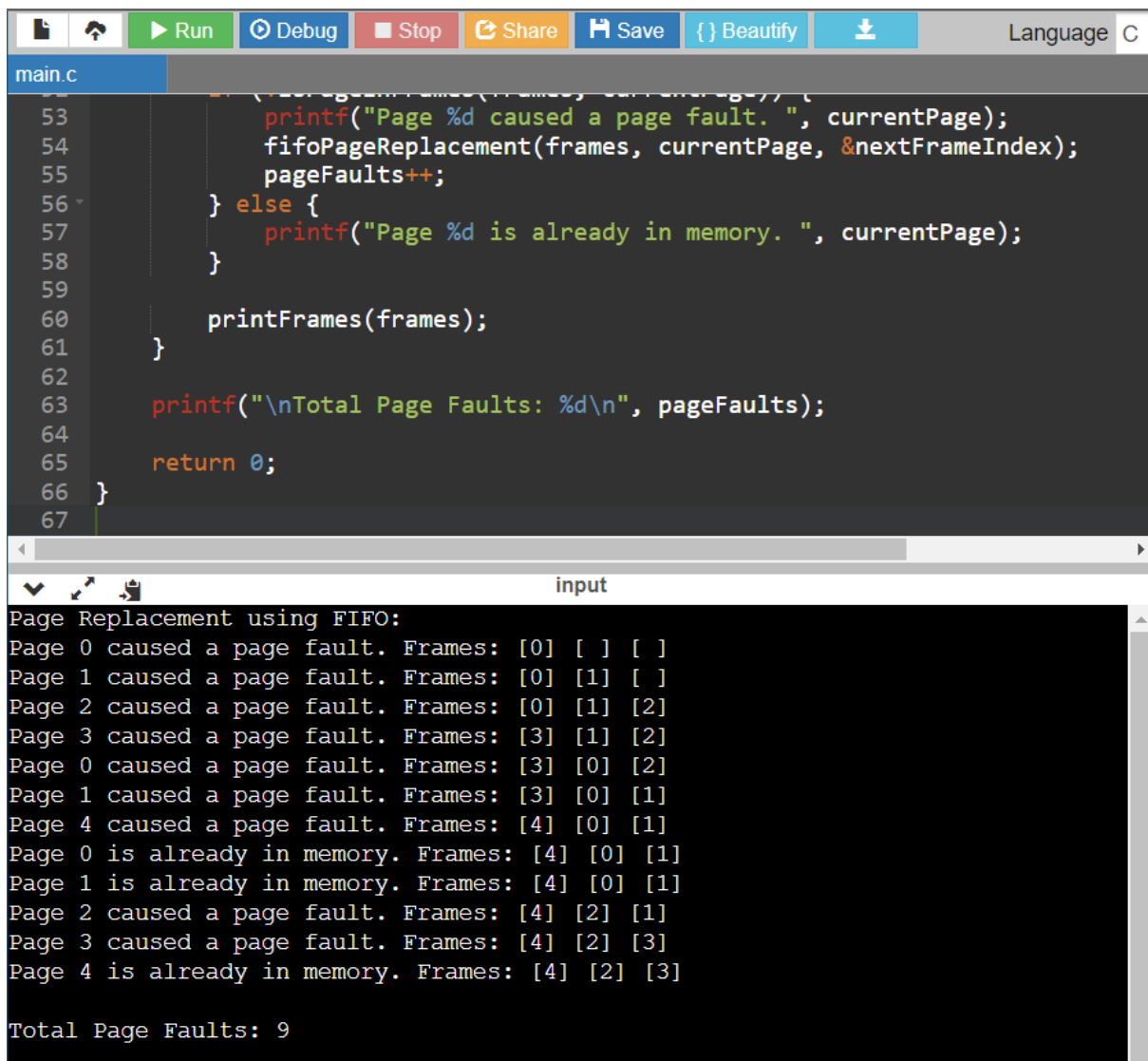
    return 0; // Page is not in frames
}

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



The screenshot shows a code editor with a C program for FIFO page replacement. The code is in a file named `main.c`. It includes a `printFrames` function and a `fifoPageReplacement` function. The `fifoPageReplacement` function checks if a page is already in memory; if not, it causes a page fault and replaces the oldest page in memory (FIFO). The program tracks the number of page faults and prints the state of memory frames after each operation.

```
main.c
53     printf("Page %d caused a page fault. ", currentPage);
54     fifoPageReplacement(frames, currentPage, &nextFrameIndex);
55     pageFaults++;
56 } else {
57     printf("Page %d is already in memory. ", currentPage);
58 }
59
60     printFrames(frames);
61 }
62
63     printf("\nTotal Page Faults: %d\n", pageFaults);
64
65     return 0;
66 }
67
```

The output of the program is shown in a terminal window titled "input". It displays the sequence of page requests and the state of the 3-frame memory at each step.

```
input
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");

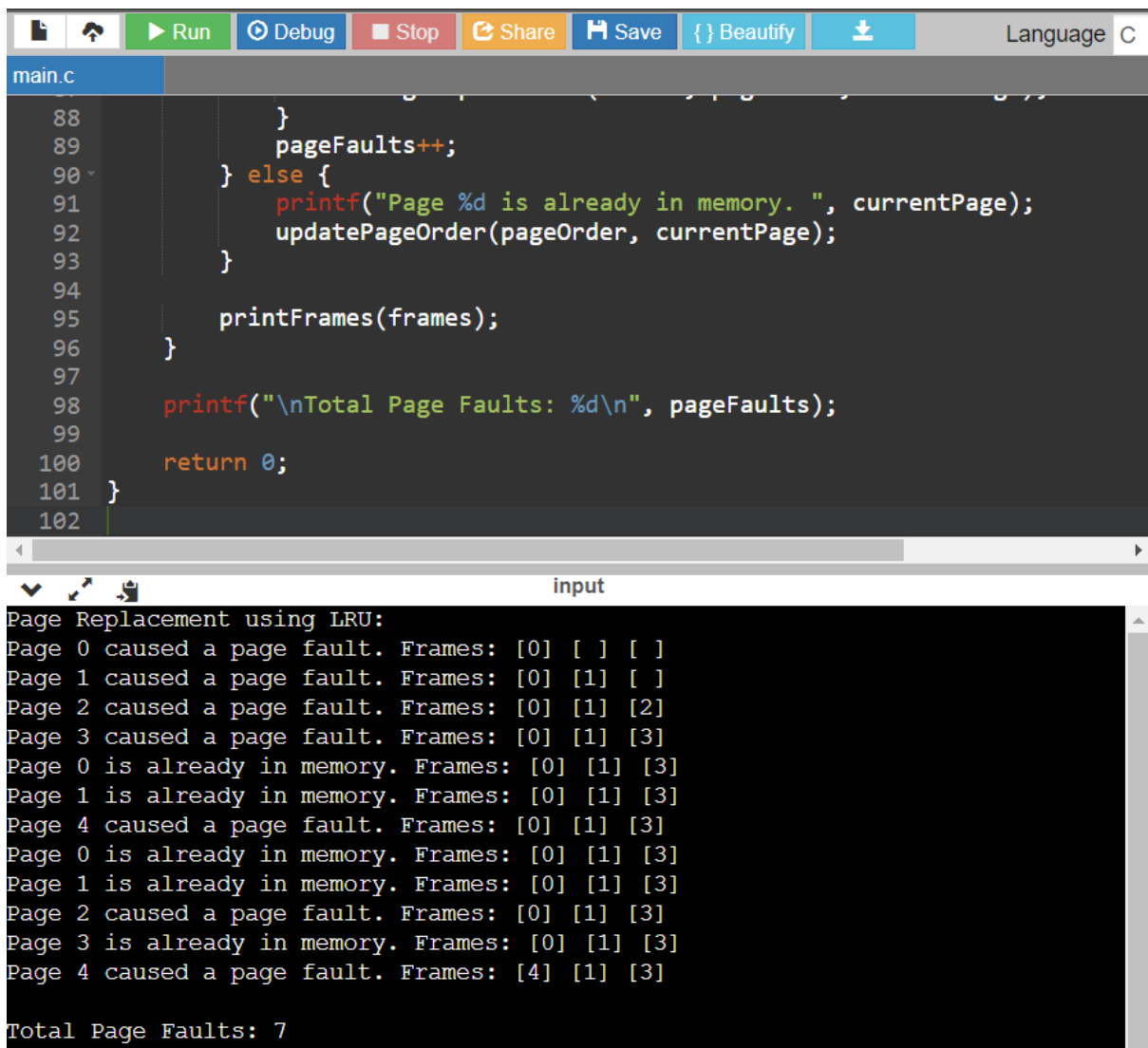
    for (int i = 0; i < pageSequenceSize; i++) {
        int currentPage = pageSequence[i];

        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



The image shows a C code editor window with a toolbar at the top containing icons for file operations, running, debugging, stopping, sharing, saving, beautifying, and downloading. The language is set to C. The code in the editor is as follows:

```
main.c
88     }
89     pageFaults++;
90 } else {
91     printf("Page %d is already in memory. ", currentPage);
92     updatePageOrder(pageOrder, currentPage);
93 }
94
95 printFrames(frames);
96 }
97
98 printf("\nTotal Page Faults: %d\n", pageFaults);
99
100 return 0;
101 }
102
```

Below the code editor is an output terminal window titled "input". It displays the following output:

```
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);
}

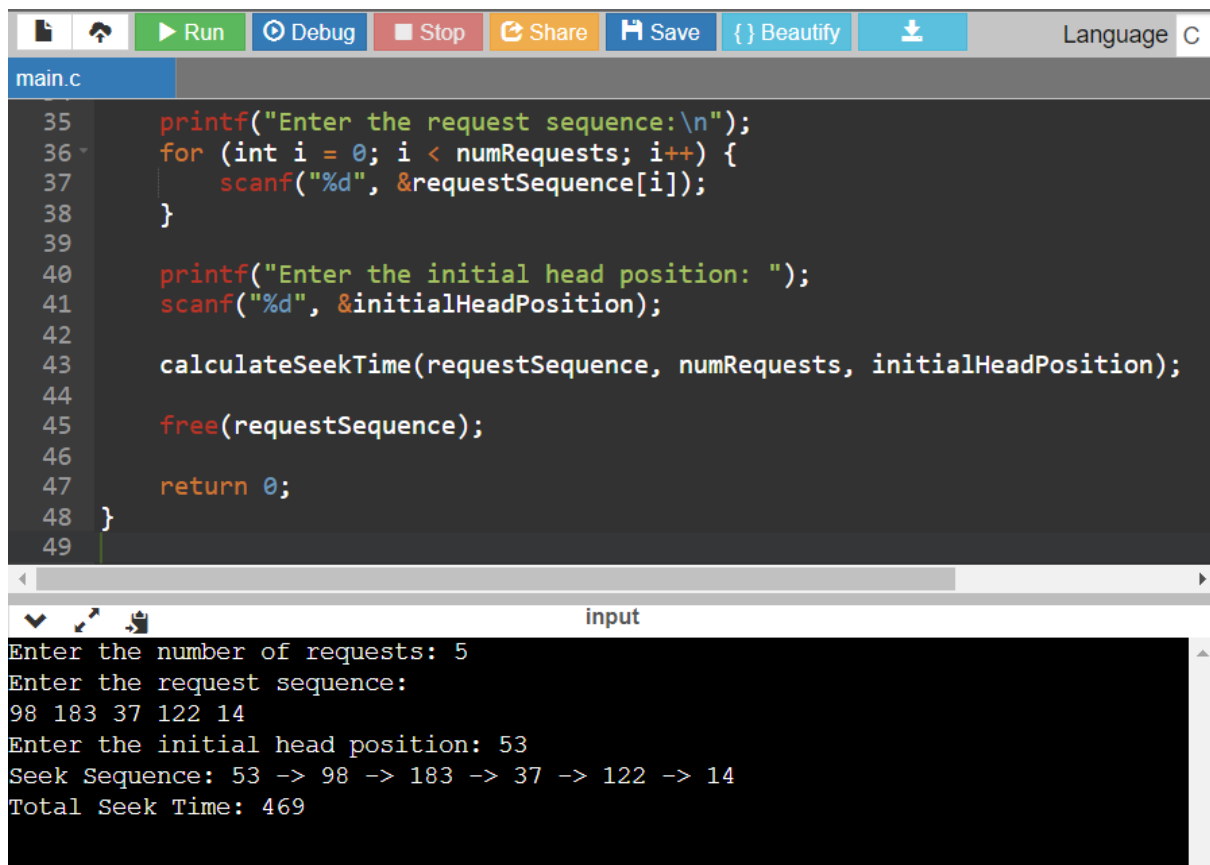
int main() {
    int numRequests;
    int initialHeadPosition;

    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



The image shows a C code editor window with a toolbar at the top containing icons for file operations, running, debugging, stopping, sharing, saving, and beautifying code. The language is set to C. The code in the editor is as follows:

```
main.c
35     printf("Enter the request sequence:\n");
36     for (int i = 0; i < numRequests; i++) {
37         scanf("%d", &requestSequence[i]);
38     }
39
40     printf("Enter the initial head position: ");
41     scanf("%d", &initialHeadPosition);
42
43     calculateSeekTime(requestSequence, numRequests, initialHeadPosition);
44
45     free(requestSequence);
46
47     return 0;
48 }
49
```

Below the code editor is a terminal window titled "input" showing the program's execution:

```
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);
```

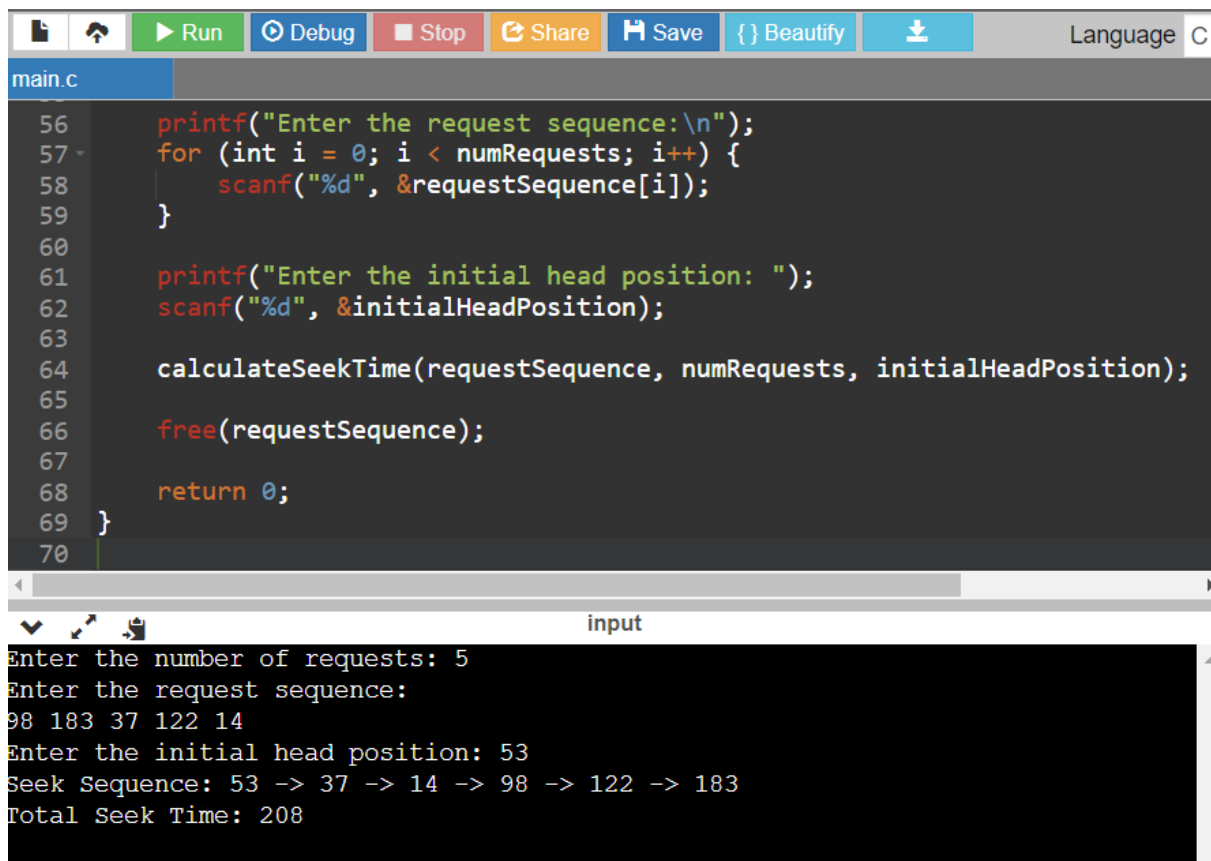
```
calculateSeekTime(requestSequence, numRequests, initialHeadPosition);
```

```
free(requestSequence);
```

```
return 0;
```

```
}
```

Output



The image shows a code editor window with a toolbar at the top containing icons for file operations, a 'Run' button, a 'Debug' button, a 'Stop' button, a 'Share' button, a 'Save' button, a 'Beautify' button, and a download icon. The language is set to 'C'. The editor shows a file named 'main.c' with the following C code:

```
56     printf("Enter the request sequence:\n");
57     for (int i = 0; i < numRequests; i++) {
58         scanf("%d", &requestSequence[i]);
59     }
60
61     printf("Enter the initial head position: ");
62     scanf("%d", &initialHeadPosition);
63
64     calculateSeekTime(requestSequence, numRequests, initialHeadPosition);
65
66     free(requestSequence);
67
68     return 0;
69 }
70
```

Below the editor is a terminal window titled 'input' showing the program's execution:

```
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));

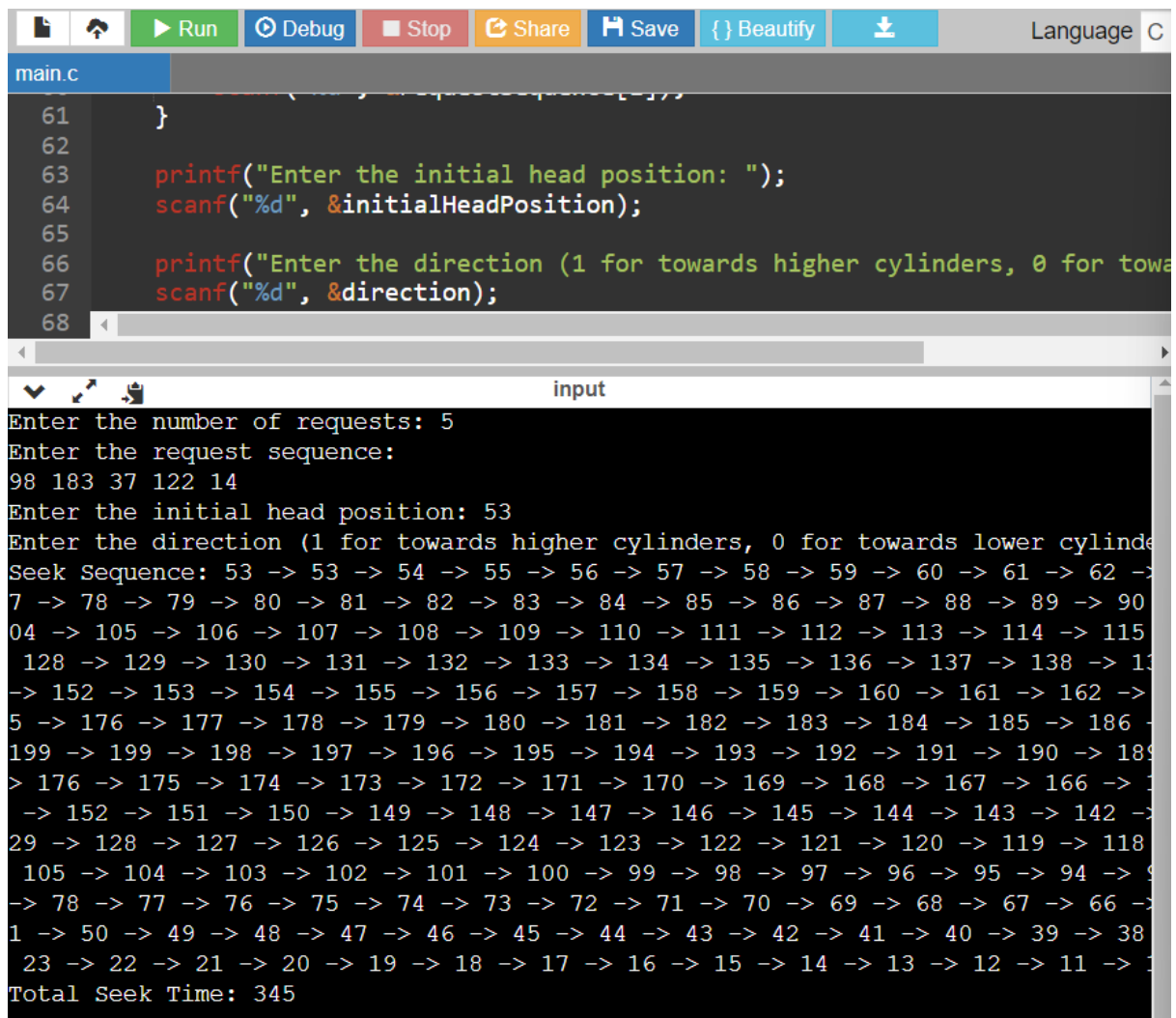
    if (requestSequence == NULL) {
        fprintf(stderr, "Memory allocation failed.\n");
        return 1; // Exit with an error code
    }

    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



The image shows a screenshot of a C program's output. The top part is a code editor window titled 'main.c' with a toolbar containing icons for Run, Debug, Stop, Share, Save, Beautify, and a download icon. The code in the editor includes prompts for the number of requests, request sequence, initial head position, and direction. The bottom part is a terminal window titled 'input' showing the program's execution. The user enters 5 requests, the sequence 98 183 37 122 14, initial head position 53, and direction 1. The program outputs a seek sequence starting at 53 and visiting all cylinders from 0 to 199 in a specific order, resulting in a total seek time of 345.

```
main.c
61 }
62
63 printf("Enter the initial head position: ");
64 scanf("%d", &initialHeadPosition);
65
66 printf("Enter the direction (1 for towards higher cylinders, 0 for towards lower cylinders)");
67 scanf("%d", &direction);
68
```

```
input
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 cylinders): 1
Seek Sequence: 53 -> 53 -> 54 -> 55 -> 56 -> 57 -> 58 -> 59 -> 60 -> 61 -> 62 -> 63 -> 64 -> 65 -> 66 -> 67 -> 68 -> 69 -> 70 -> 71 -> 72 -> 73 -> 74 -> 75 -> 76 -> 77 -> 78 -> 79 -> 80 -> 81 -> 82 -> 83 -> 84 -> 85 -> 86 -> 87 -> 88 -> 89 -> 90 -> 91 -> 92 -> 93 -> 94 -> 95 -> 96 -> 97 -> 98 -> 99 -> 100 -> 101 -> 102 -> 103 -> 104 -> 105 -> 106 -> 107 -> 108 -> 109 -> 110 -> 111 -> 112 -> 113 -> 114 -> 115 -> 116 -> 117 -> 118 -> 119 -> 120 -> 121 -> 122 -> 123 -> 124 -> 125 -> 126 -> 127 -> 128 -> 129 -> 130 -> 131 -> 132 -> 133 -> 134 -> 135 -> 136 -> 137 -> 138 -> 139 -> 140 -> 141 -> 142 -> 143 -> 144 -> 145 -> 146 -> 147 -> 148 -> 149 -> 150 -> 151 -> 152 -> 153 -> 154 -> 155 -> 156 -> 157 -> 158 -> 159 -> 160 -> 161 -> 162 -> 163 -> 164 -> 165 -> 166 -> 167 -> 168 -> 169 -> 170 -> 171 -> 172 -> 173 -> 174 -> 175 -> 176 -> 177 -> 178 -> 179 -> 180 -> 181 -> 182 -> 183 -> 184 -> 185 -> 186 -> 187 -> 188 -> 189 -> 190 -> 191 -> 192 -> 193 -> 194 -> 195 -> 196 -> 197 -> 198 -> 199 -> 199 -> 198 -> 197 -> 196 -> 195 -> 194 -> 193 -> 192 -> 191 -> 190 -> 189 -> 188 -> 187 -> 186 -> 185 -> 184 -> 183 -> 182 -> 181 -> 180 -> 179 -> 178 -> 177 -> 176 -> 175 -> 174 -> 173 -> 172 -> 171 -> 170 -> 169 -> 168 -> 167 -> 166 -> 165 -> 164 -> 163 -> 162 -> 161 -> 160 -> 159 -> 158 -> 157 -> 156 -> 155 -> 154 -> 153 -> 152 -> 151 -> 150 -> 149 -> 148 -> 147 -> 146 -> 145 -> 144 -> 143 -> 142 -> 141 -> 140 -> 139 -> 138 -> 137 -> 136 -> 135 -> 134 -> 133 -> 132 -> 131 -> 130 -> 129 -> 128 -> 127 -> 126 -> 125 -> 124 -> 123 -> 122 -> 121 -> 120 -> 119 -> 118 -> 117 -> 116 -> 115 -> 114 -> 113 -> 112 -> 111 -> 110 -> 109 -> 108 -> 107 -> 106 -> 105 -> 104 -> 103 -> 102 -> 101 -> 100 -> 99 -> 98 -> 97 -> 96 -> 95 -> 94 -> 93 -> 92 -> 91 -> 90 -> 89 -> 88 -> 87 -> 86 -> 85 -> 84 -> 83 -> 82 -> 81 -> 80 -> 79 -> 78 -> 77 -> 76 -> 75 -> 74 -> 73 -> 72 -> 71 -> 70 -> 69 -> 68 -> 67 -> 66 -> 65 -> 64 -> 63 -> 62 -> 61 -> 60 -> 59 -> 58 -> 57 -> 56 -> 55 -> 54 -> 53 -> 52 -> 51 -> 50 -> 49 -> 48 -> 47 -> 46 -> 45 -> 44 -> 43 -> 42 -> 41 -> 40 -> 39 -> 38 -> 37 -> 36 -> 35 -> 34 -> 33 -> 32 -> 31 -> 30 -> 29 -> 28 -> 27 -> 26 -> 25 -> 24 -> 23 -> 22 -> 21 -> 20 -> 19 -> 18 -> 17 -> 16 -> 15 -> 14 -> 13 -> 12 -> 11 -> 10 -> 9 -> 8 -> 7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1 -> 0
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]);
    }

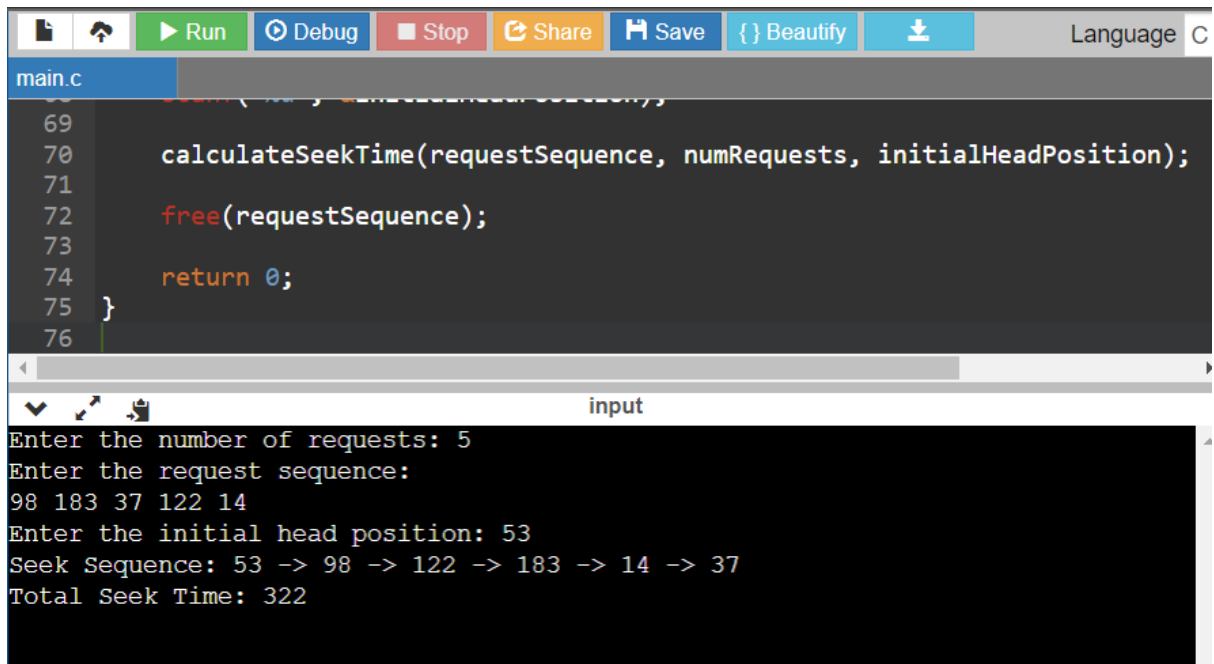
    printf("Enter the initial head position: ");
    scanf("%d", &initialHeadPosition);

    calculateSeekTime(requestSequence, numRequests, initialHeadPosition);

    free(requestSequence);

    return 0;
}
```

Output



The image shows a screenshot of a C program being executed in an IDE. The top toolbar includes buttons for Run, Debug, Stop, Share, Save, Beautify, and a download icon. The language is set to C. The code in main.c is as follows:

```
69  
70     calculateSeekTime(requestSequence, numRequests, initialHeadPosition);  
71  
72     free(requestSequence);  
73  
74     return 0;  
75 }  
76
```

Below the code editor is a terminal window titled "input" showing the program's execution:

```
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
```