PROGRAM-Implementation of FIFO page replacement algorithms.

```
#include <stdio.h>
#include <stdbool.h>
// Function to check if a page exists in a frame
bool isInFrame(int page, int frames[], int frameSize) {
    for (int i = 0; i < frameSize; i++) {
        if (frames[i] == page)
            return true;
    }
    return false;
}
// Function to find the index of the page that entered first (FIFO)
int findFIFOIndex(int pages[], int frames[], int frameSize, int currentPageIndex) {
    int oldestPageIndex = currentPageIndex;
    for (int i = 0; i < frameSize; i++) {
        int j;
        for (j = currentPageIndex - 1; j >= 0; j--) {
            if (frames[i] == pages[j]) {
                if (j < oldestPageIndex) {
                    oldestPageIndex = j;
                }
                break;
            }
        }
        if (j == -1)
            return i;
    }
    return oldestPageIndex;
}

// Function to display the frames
void displayFrames(int frames[], int frameSize) {
    for (int i = 0; i < frameSize; i++) {
        if (frames[i] == -1)
            printf("X ");
        else
            printf("%d ", frames[i]);
    }
    printf("\n");
}

// FIFO Page Replacement Algorithm
void fifo(int pages[], int numPages, int frameSize) {
    int frames[frameSize];
    int pageFaults = 0;
    int frameIndex = 0;
    for (int i = 0; i < frameSize; i++)
        frames[i] = -1;
    for (int i = 0; i < numPages; i++) {
        if (!isInFrame(pages[i], frames, frameSize)) {
            pageFaults++;
            frames[frameIndex] = pages[i];
            frameIndex = (frameIndex + 1) % frameSize;
        }
        printf("Page %d: ", pages[i]);
        displayFrames(frames, frameSize);
    }
    printf("\nTotal Page Faults: %d\n", pageFaults);
}
int main() {
    int numPages, frameSize;
    printf("Enter the number of pages: ");
    scanf("%d", &numPages);
    int pages[numPages];
    printf("Enter the page references: ");
    for (int i = 0; i < numPages; i++) {
        scanf("%d", &pages[i]);
    }
    printf("Enter the frame size: ");
    scanf("%d", &frameSize);
    fifo(pages, numPages, frameSize);
    return 0;
}
```

OUTPUT:

```
Enter the number of pages: 7
Enter the page references:
1
3
0
3
5
6
3
Enter the frame size: 3
Page 1: 1 X X
Page 3: 1 3 X
Page 0: 1 3 0
Page 3: 1 3 0
Page 5: 5 3 0
Page 6: 5 6 0
Page 3: 5 6 3

Total Page Faults: 6
```

PROGRAM: Implementation Of FCFS Disk Scheduling Algorithm.

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

// Function to calculate the total head movement
int calculateTotalHeadMovement(int *requests, int numRequests, int initialPosition) {
    int totalHeadMovement = 0;

    // Iterate through the requests and calculate head movement
    for (int i = 0; i < numRequests; i++) {
        // Calculate absolute difference between current request and previous request
        int movement = abs(requests[i] - initialPosition);
        // Add it to total head movement
        totalHeadMovement += movement;
        // Update initial position
        initialPosition = requests[i];
    }

    return totalHeadMovement;
}

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

    int requests[numRequests];
    printf("Enter the disk requests: ");
    for (int i = 0; i < numRequests; i++) {
        scanf("%d", &requests[i]);
    }

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

    // Calculate total head movement using FCFS algorithm
    int totalHeadMovement = calculateTotalHeadMovement(requests, numRequests,
initialPosition);
    printf("Total head movement: %d\n", totalHeadMovement);

    return 0;
}
```

OUTPUT:

```
Enter the number of disk requests: 8
Enter the disk requests:
176
79
34
60
92
11
41
114
Enter the initial head position: 50
Total head movement: 510
```

PROGRAM: Implementation of various memory allocation algorithms, (First fit, best fit and Worst fit).

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

#define MAX_BLOCKS 100

// Structure to represent a memory block
typedef struct {
    int id;
    int size;
    int allocated;
} MemoryBlock;

// Function prototypes
void firstFit(MemoryBlock blocks[], int numBlocks, int requestSize);
void bestFit(MemoryBlock blocks[], int numBlocks, int requestSize);
void worstFit(MemoryBlock blocks[], int numBlocks, int requestSize);
void printBlocks(MemoryBlock blocks[], int numBlocks, const char* allocationAlgorithm);

int main() {
    MemoryBlock blocks[MAX_BLOCKS];
    int numBlocks, i, requestSize;

    printf("Enter the number of memory blocks: ");
    scanf("%d", &numBlocks);

    // Input memory blocks
    for (i = 0; i < numBlocks; i++) {
        printf("Enter size of block %d: ", i + 1);
        scanf("%d", &blocks[i].size);
        blocks[i].id = i + 1;
        blocks[i].allocated = 0;
    }

    // Print initial state of memory blocks
    printf("\nInitial state of memory blocks:\n");
    printBlocks(blocks, numBlocks, "None");

    // Process memory allocation requests
    printf("\nEnter size of memory request: ");
    scanf("%d", &requestSize);

    printf("\nFirst Fit Allocation:\n");
    firstFit(blocks, numBlocks, requestSize);
    printBlocks(blocks, numBlocks, "First Fit");

    printf("\nBest Fit Allocation:\n");
    bestFit(blocks, numBlocks, requestSize);
    printBlocks(blocks, numBlocks, "Best Fit");

    printf("\nWorst Fit Allocation:\n");
    worstFit(blocks, numBlocks, requestSize);
    printBlocks(blocks, numBlocks, "Worst Fit");

    return 0;
}

// First Fit Allocation Algorithm
void firstFit(MemoryBlock blocks[], int numBlocks, int requestSize) {
    int i;
    for (i = 0; i < numBlocks; i++) {
        if (blocks[i].allocated == 0 && blocks[i].size >= requestSize) {
            blocks[i].allocated = 1;
            break;
        }
    }
}

// Best Fit Allocation Algorithm
void bestFit(MemoryBlock blocks[], int numBlocks, int requestSize) {
    int i, bestFitIndex = -1, minFragmentation = INT_MAX;
    for (i = 0; i < numBlocks; i++) {
        if (blocks[i].allocated == 0 && blocks[i].size >= requestSize) {
            int fragmentation = blocks[i].size - requestSize;
            if (fragmentation < minFragmentation) {
                minFragmentation = fragmentation;
                bestFitIndex = i;
            }
        }
    }
    if (bestFitIndex != -1)
        blocks[bestFitIndex].allocated = 1;
}

// Worst Fit Allocation Algorithm
void worstFit(MemoryBlock blocks[], int numBlocks, int requestSize) {
    int i, worstFitIndex = -1, maxFragmentation = -1;
    for (i = 0; i < numBlocks; i++) {
        if (blocks[i].allocated == 0 && blocks[i].size >= requestSize) {
            int fragmentation = blocks[i].size - requestSize;
            if (fragmentation > maxFragmentation) {
                maxFragmentation = fragmentation;
                worstFitIndex = i;
            }
        }
    }
```

```
if (worstFitIndex != -1)
        blocks[worstFitIndex].allocated = 1;
}

// Function to print memory blocks
void printBlocks(MemoryBlock blocks[], int numBlocks, const char* allocationAlgorithm) {
    int i;
    printf("Block\tSize\tAllocated (%s)\n", allocationAlgorithm);
    for (i = 0; i < numBlocks; i++) {
        printf("%d\t%d\t%s\n", blocks[i].id, blocks[i].size, blocks[i].allocated ? "Yes" : "No");
    }
}
```

OUTPUT:

```
Enter the number of memory blocks: 5
Enter size of block 1: 20
Enter size of block 2: 15
Enter size of block 3: 25
Enter size of block 4: 30
Enter size of block 5: 40

Initial state of memory blocks:
Block   Size    Allocated (None)
1       20      No
2       15      No
3       25      No
4       30      No
5       40      No

Enter size of memory request: 18

First Fit Allocation:
Block   Size    Allocated (First Fit)
1       20      Yes
2       15      No
3       25      No
4       30      No
5       40      No

Best Fit Allocation:
Block   Size    Allocated (Best Fit)
1       20      Yes
2       15      No
3       25      Yes
4       30      No
5       40      No
```

Worst Fit Allocation:

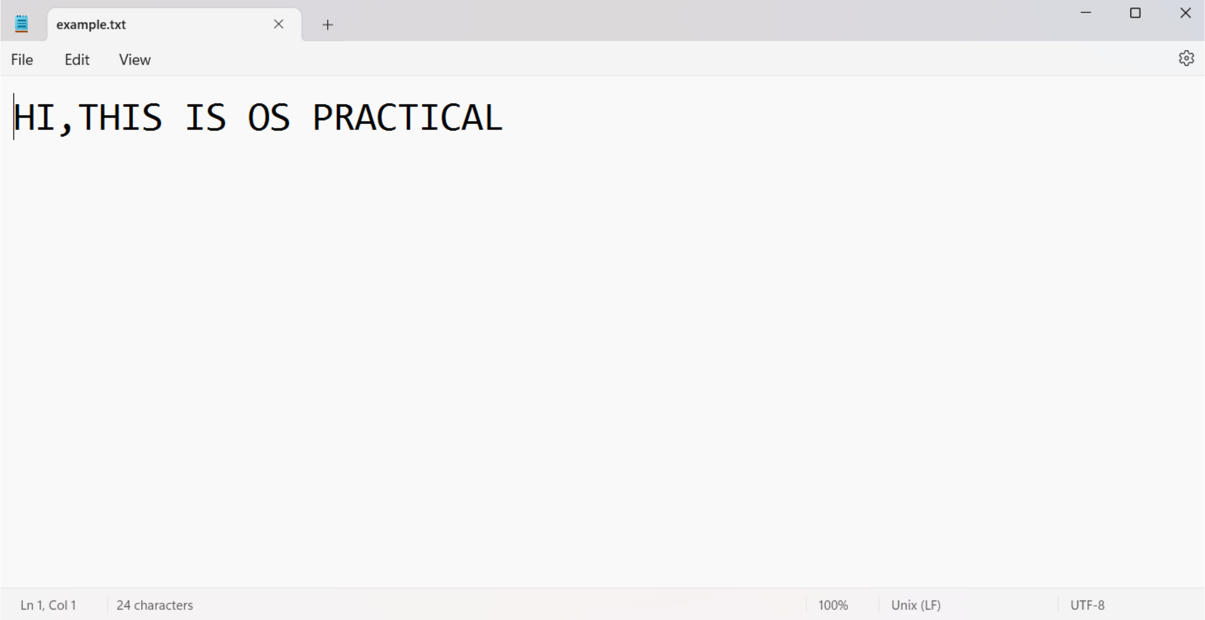
Block	Size	Allocated (Worst Fit)
1	20	Yes
2	15	No
3	25	Yes
4	30	No
5	40	Yes

PROGRAM: Write A Program Illustrating Various File Handling Functions.

```
#include <stdio.h>

int main()
{
    FILE *filePointer;
    char fileName[] = "example.txt";
    char content[] = "HI,THIS IS ANKITA.\n";
    char buffer[100];

    // Open file for writing
    filePointer = fopen(fileName, "w");
    if (filePointer == NULL) {
        printf("Error opening file for writing.\n");
        return 1;
    }

    // Write content to the file
    fprintf(filePointer, "%s", content);

    // Close the file
    fclose(filePointer);

    printf("File created and content written successfully.\n");

    // Open file for reading
    filePointer = fopen(fileName, "r");
    if (filePointer == NULL) {
        printf("Error opening file for reading.\n");
        return 1;
    }

    // Read content from the file
    printf("Contents of the file:\n");
    while (fgets(buffer, sizeof(buffer), filePointer) != NULL) {
        printf("%s", buffer);
    }

    // Close the file
    fclose(filePointer);

    return 0;
}
```

OUTPUT:

```
File created and content written successfully.
Contents of the file:
HI,THIS IS OS PRACTICAL.
```



```
HI,THIS IS OS PRACTICAL
```

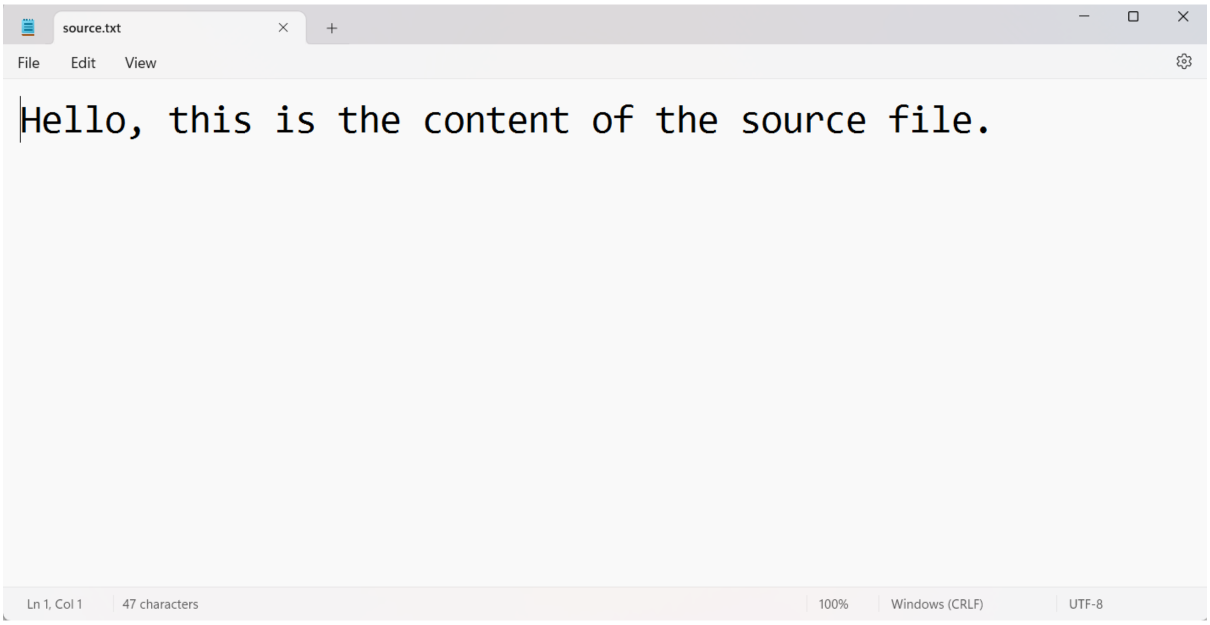
PROGRAM: Write A Program For Copying Content Of One File To Other.

```
#include <stdio.h>
#include <stdlib.h>
#define BUFFER_SIZE 4096

int main()
{
   FILE *sourceFile, *destinationFile;
   char buffer[BUFFER_SIZE];
   size_t bytesRead;

   // Open the source file for reading
   sourceFile = fopen("source.txt", "rb");
   if (sourceFile == NULL)
  {
      printf("Error opening source file\n");
      return EXIT_FAILURE;
   }

   // Open the destination file for writing
   destinationFile = fopen("destination.txt", "wb");
   if (destinationFile == NULL)
   {
      printf("Error opening destination file\n");
      fclose(sourceFile);
      return EXIT_FAILURE;
   }

   // Copying content from source to destination
   while ((bytesRead = fread(buffer, 1, BUFFER_SIZE, sourceFile)) > 0) {
      fwrite(buffer, 1, bytesRead, destinationFile);
   }

   // Check for errors during copying
   if (ferror(sourceFile)) {
      printf("Error reading from source file\n");
      fclose(sourceFile);
      fclose(destinationFile);
      return EXIT_FAILURE;
   }
   if (ferror(destinationFile))
{
      printf("Error writing to destination file\n");
      fclose(sourceFile);
      fclose(destinationFile);
      return EXIT_FAILURE;
   }
```

```
    // Close the files
    fclose(sourceFile);
    fclose(destinationFile);

printf("File copied successfully.\n");
return EXIT_SUCCESS;
}
```

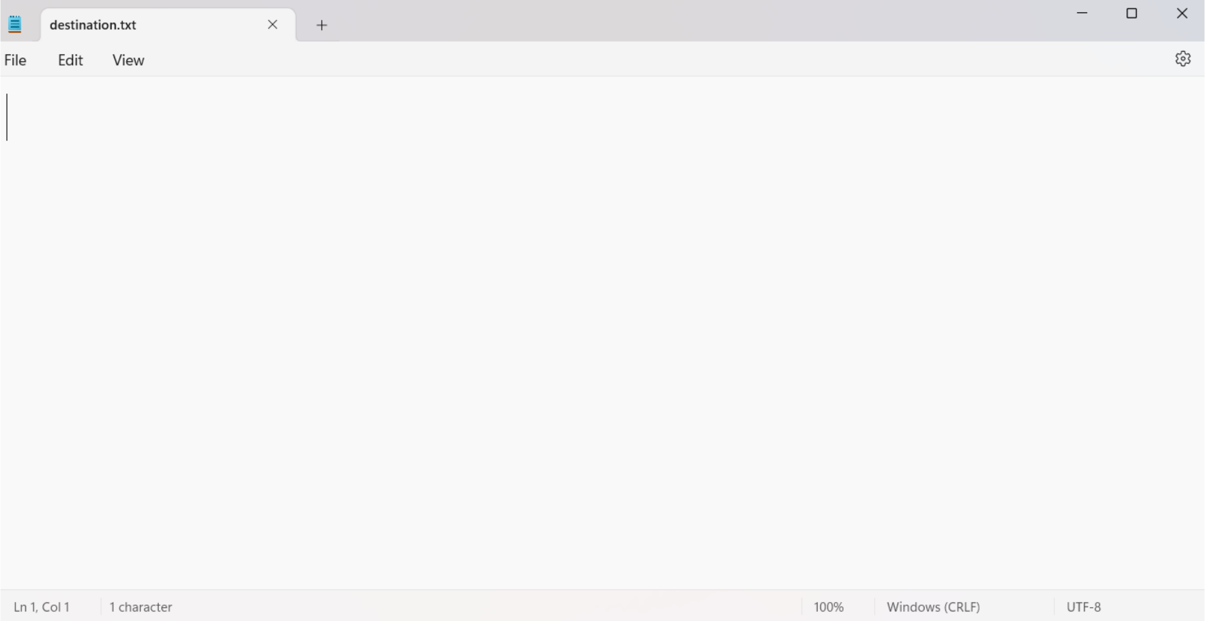
OUTPUT:

Source.txt



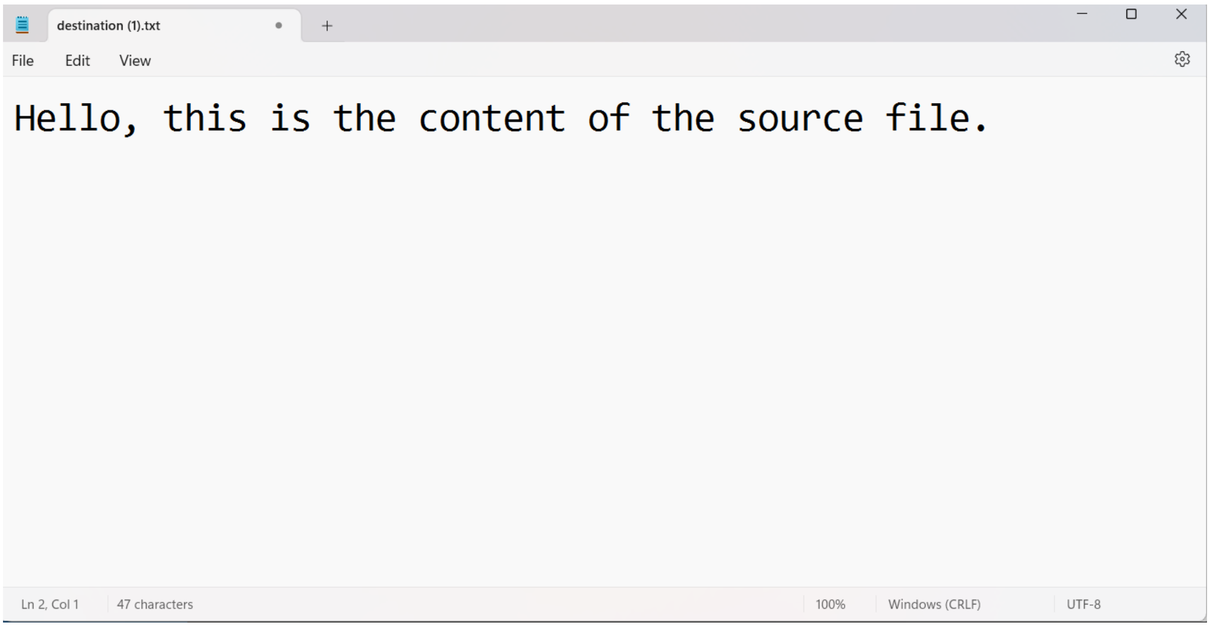
```
Hello, this is the content of the source file.
```

Destination.txt



File copied successfully.

Destination.txt



```
Hello, this is the content of the source file.
```