Programiz

C Online Compiler

main.c

Share | Run

```c
#include <stdio.h>

#define MAX 100

int main() {
    int blockSize[MAX], processSize[MAX];
    int blockCount, processCount;
    int allocation[MAX]; // stores index of block allocated to
        process

    // Input number of blocks and processes
    printf("Enter number of memory blocks: ");
    scanf("%d", &blockCount);
    printf("Enter number of processes: ");
    scanf("%d", &processCount);

    // Input sizes of memory blocks
    printf("Enter sizes of memory blocks:\n");
    for (int i = 0; i < blockCount; i++) {
        printf("Block %d: ", i + 1);
        scanf("%d", &blockSize[i]);
    }

    // Input sizes of processes
    printf("Enter sizes of processes:\n");
    for (int i = 0; i < processCount; i++) {
```

Output | Clear

```
Enter number of memory blocks: 4
Enter number of processes: 8
Enter sizes of memory blocks:
Block 1: 100
Block 2: 500
Block 3: 300
Block 4: 200
Enter sizes of processes:
Process 1: 212
Process 2: 451
Process 3: 621
Process 4: 751
Process 5: 321
Process 6: 235
Process 7: 263
Process 8: 123

Process No. Process Size    Block No.
1           212             2
2           451             Not Allocated
3           621             Not Allocated
4           751             Not Allocated
5           321             Not Allocated
6           235             3
7           263             2
8           123             4
```

main.c

Share Run

```
26          printf("Process %d: ", i + 1);
27          scanf("%d", &processSize[i]);
28      }
29
30      // Initialize allocation to -1 (not allocated)
31      for (int i = 0; i < processCount; i++)
32          allocation[i] = -1;
33
34      // Worst Fit Allocation
35      for (int i = 0; i < processCount; i++) {
36          int worstIdx = -1;
37          for (int j = 0; j < blockCount; j++) {
38              if (blockSize[j] >= processSize[i]) {
39                  if (worstIdx == -1 || blockSize[j] >
                        blockSize[worstIdx])
40                      worstIdx = j;
41              }
42          }
43
44          if (worstIdx != -1) {
45              allocation[i] = worstIdx;
46              blockSize[worstIdx] -= processSize[i];
47          }
48      }
49
50      // Output allocation result
```

Output

```
Block 1: 100
Block 2: 500
Block 3: 300
Block 4: 200
Enter sizes of processes:
Process 1: 212
Process 2: 451
Process 3: 621
Process 4: 751
Process 5: 321
Process 6: 235
Process 7: 263
Process 8: 123

Process No.  Process Size    Block No.
1            212             2
2            451             Not Allocated
3            621             Not Allocated
4            751             Not Allocated
5            321             Not Allocated
6            235             3
7            263             2
8            123             4

=== Code Execution Successful ===
```

main.c

Share Run

```c
            if (blockSize[j] >= processSize[i]) {
                if (worstIdx == -1 || blockSize[j] >
                    blockSize[worstIdx])
                    worstIdx = j;
            }
        }

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

    // Output allocation result
    printf("\nProcess No.\tProcess Size\tBlock No.\n");
    for (int i = 0; i < processCount; 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

```
Block 1: 100
Block 2: 500
Block 3: 300
Block 4: 200
Enter sizes of processes:
Process 1: 212
Process 2: 451
Process 3: 621
Process 4: 751
Process 5: 321
Process 6: 235
Process 7: 263
Process 8: 123

Process No.  Process Size    Block No.
1            212             2
2            451             Not Allocated
3            621             Not Allocated
4            751             Not Allocated
5            321             Not Allocated
6            235             3
7            263             2
8            123             4


=== Code Execution Successful ===
```

Programiz

C Online Compiler

main.c

Share Run

```c
#include <stdio.h>

#define MAX 100

int main() {
    int blockSize[MAX], processSize[MAX];
    int blockCount, processCount;
    int allocation[MAX]; // stores block index allocated to process

    // Input number of blocks and processes
    printf("Enter number of memory blocks: ");
    scanf("%d", &blockCount);
    printf("Enter number of processes: ");
    scanf("%d", &processCount);

    // Input sizes of memory blocks
    printf("Enter sizes of memory blocks:\n");
    for (int i = 0; i < blockCount; i++) {
        printf("Block %d: ", i + 1);
        scanf("%d", &blockSize[i]);
    }

    // Input sizes of processes
    printf("Enter sizes of processes:\n");
    for (int i = 0; i < processCount; i++) {
        printf("Process %d: ", i + 1);
```

Output

```
Enter number of memory blocks: 4
Enter number of processes: 5
Enter sizes of memory blocks:
Block 1: 100
Block 2: 500
Block 3: 600
Block 4: 700
Enter sizes of processes:
Process 1: 212
Process 2: 471
Process 3: 523
Process 4: 702
Process 5: 902

Process No.  Process Size    Block No.
1            212             2
2            471             3
3            523             4
4            702             Not Allocated
5            902             Not Allocated

=== Code Execution Successful ===
```

main.c

Online Python Compiler

Share | Run

```
38          for (int j = 0; j < blockCount; j++) {
39              if (blockSize[j] >= processSize[i]) {
40                  if (bestIdx == -1 || blockSize[j] <
                        blockSize[bestIdx])
41                      bestIdx = j;
42              }
43          }
44
45          if (bestIdx != -1) {
46              allocation[i] = bestIdx;
47              blockSize[bestIdx] -= processSize[i];
48          }
49      }
50
51      // Output allocation result
52      printf("\nProcess No.\tProcess Size\tBlock No.\n");
53      for (int i = 0; i < processCount; i++) {
54          printf("%d\t\t%d\t\t", i + 1, processSize[i]);
55          if (allocation[i] != -1)
56              printf("%d\n", allocation[i] + 1);
57          else
58              printf("Not Allocated\n");
59      }
60
61      return 0;
```

Output

```
Enter number of memory blocks: 4
Enter number of processes: 5
Enter sizes of memory blocks:
Block 1: 100
Block 2: 500
Block 3: 600
Block 4: 700
Enter sizes of processes:
Process 1: 212
Process 2: 471
Process 3: 523
Process 4: 702
Process 5: 902

Process No.  Process Size    Block No.
1            212             2
2            471             3
3            523             4
4            702             Not Allocated
5            902             Not Allocated


=== Code Execution Successful ===
```

https://www.programiz.com/python-programming/online-compiler/

Programiz
C Online Compiler

main.c

Share | Run | Output | Clear

```c
#include <stdio.h>

#define MAX 100

int main() {
    int blockSize[MAX], processSize[MAX];
    int blockCount, processCount;
    int allocation[MAX]; // stores block index allocated to process

    // Input number of blocks and processes
    printf("Enter number of memory blocks: ");
    scanf("%d", &blockCount);
    printf("Enter number of processes: ");
    scanf("%d", &processCount);

    // Input sizes of memory blocks
    printf("Enter sizes of memory blocks:\n");
    for (int i = 0; i < blockCount; i++) {
        printf("Block %d: ", i + 1);
        scanf("%d", &blockSize[i]);
    }

    // Input sizes of processes
    printf("Enter sizes of processes:\n");
    for (int i = 0; i < processCount; i++) {
        printf("Process %d: ", i + 1);
```

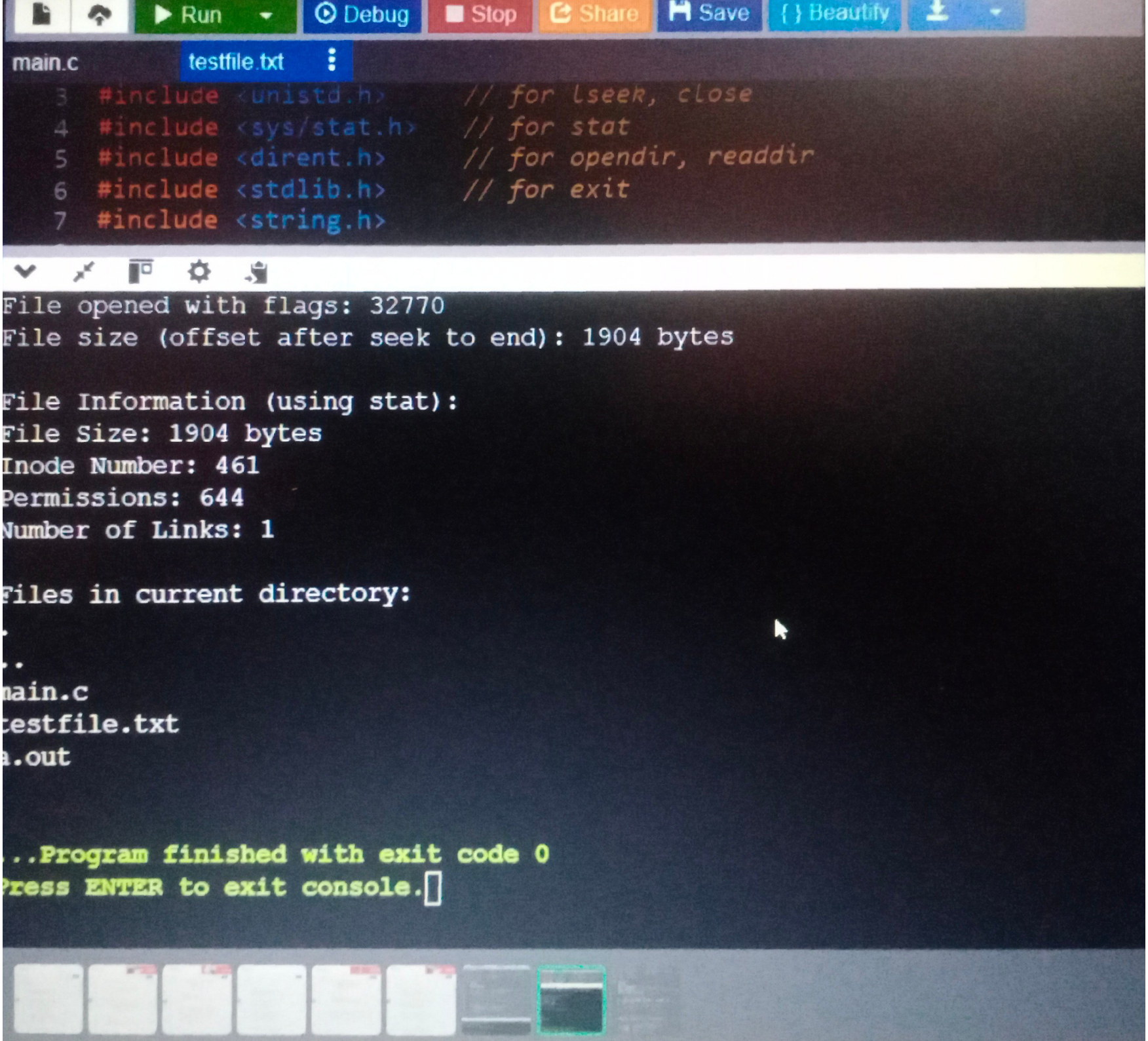
```
Enter number of memory blocks: 3
Enter number of processes: 4
Enter sizes of memory blocks:
Block 1: 500
Block 2: 100
Block 3: 600
Enter sizes of processes:
Process 1: 420
Process 2: 212
Process 3: 521
Process 4: 601

Process No. Process Size    Block No.
1           420             1
2           212             3
3           521             Not Allocated
4           601             Not Allocated


=== Code Execution Successful ===
```

Search

ENG
IN

```c
#include <fcntl.h>   // for open()
#include <unistd.h>  // for read(), write(), close(), lseek()
#include <string.h>  // for strlen()
#include <stdlib.h>  // for exit()

int main() {
    int fd;
    char buffer[100];
    ssize_t bytesRead;

    // 1. Create and open a file (with read-write permission)
    fd = open("sample.txt", O_CREAT | O_RDWR, 0644);
    if (fd < 0) {
        perror("Error creating file");
        exit(1);
    }
    printf("File 'sample.txt' created and opened successfully.\n");

    // 2. Write to the file
    char data[] = "This is a UNIX system call demo.\n";
    if (write(fd, data, strlen(data)) < 0) {
        perror("Error writing to file");
        close(fd);
        exit(1);
    }
    printf("Data written to file.\n");
```

```
Reading data from file:
This is a UNIX system call demo.
File closed successfully.
File 'sample.txt' deleted successfully.


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


Run
Debug
Stop
Share
Save
{ } Beautify
main.c
testfile.txt
#include <unistd.h> // for lseek, close
#include <sys/stat.h> // for stat
#include <dirent.h> // for opendir, readdir
#include <stdlib.h> // for exit
#include <string.h>
File opened with flags: 32770
File size (offset after seek to end): 1904 bytes
File Information (using stat):
File Size: 1904 bytes
Inode Number: 461
Permissions: 644
Number of Links: 1
Files in current directory:
.
..
main.c
testfile.txt
a.out
..Program finished with exit code 0
Press ENTER to exit console.