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

#define MEMORY_SIZE 100
#define NUM_BLOCKS 10

int memory[MEMORY_SIZE];

// Structure to represent a memory block
typedef struct {
    int starting_address;
    int size;
    int allocated;
} Block;

// Function to initialize memory blocks
void initialize_memory(Block blocks[]) {
    int block_size = MEMORY_SIZE / NUM_BLOCKS;
    for (int i = 0; i < NUM_BLOCKS; i++) {
        blocks[i].starting_address = i * block_size;
        blocks[i].size = block_size;
        blocks[i].allocated = 0;
    }
}

// First Fit Allocation Strategy
void first_fit(Block blocks[], int size) {
    for (int i = 0; i < NUM_BLOCKS; i++) {
        if (blocks[i].allocated == 0 && blocks[i].size >= size) {
            blocks[i].allocated = 1;
            printf("First Fit: Allocated Block %d at address %d\n", i, blocks[i].starting_address);
```

```
            return;
        }
    }
    printf("First Fit: Failed to allocate memory for size %d\n", size);
}

// Best Fit Allocation Strategy
void best_fit(Block blocks[], int size) {
    int best_fit_index = -1;
    int min_fragmentation = MEMORY_SIZE + 1;
    for (int i = 0; i < NUM_BLOCKS; i++) {
        if (blocks[i].allocated == 0 && blocks[i].size >= size) {
            int fragmentation = blocks[i].size - size;
            if (fragmentation < min_fragmentation) {
                min_fragmentation = fragmentation;
                best_fit_index = i;
            }
        }
    }
    if (best_fit_index != -1) {
        blocks[best_fit_index].allocated = 1;
        printf("Best Fit: Allocated Block %d at address %d\n", best_fit_index,
blocks[best_fit_index].starting_address);
    } else {
        printf("Best Fit: Failed to allocate memory for size %d\n", size);
    }
}

// Worst Fit Allocation Strategy
void worst_fit(Block blocks[], int size) {
    int worst_fit_index = -1;
```

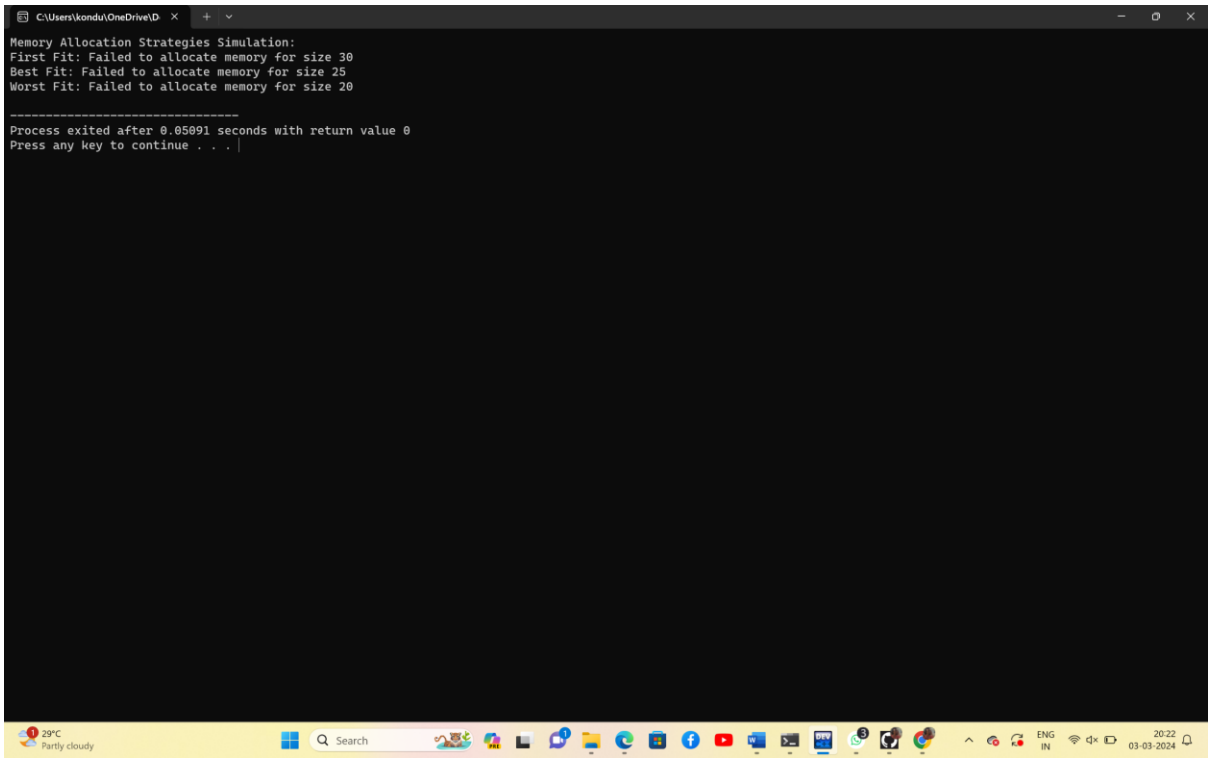
```
    int max_fragmentation = -1;
    for (int i = 0; i < NUM_BLOCKS; i++) {
        if (blocks[i].allocated == 0 && blocks[i].size >= size) {
            int fragmentation = blocks[i].size - size;
            if (fragmentation > max_fragmentation) {
                max_fragmentation = fragmentation;
                worst_fit_index = i;
            }
        }
    }
    if (worst_fit_index != -1) {
        blocks[worst_fit_index].allocated = 1;
        printf("Worst Fit: Allocated Block %d at address %d\n", worst_fit_index,
blocks[worst_fit_index].starting_address);
    } else {
        printf("Worst Fit: Failed to allocate memory for size %d\n", size);
    }
}

int main() {
    Block blocks[NUM_BLOCKS];
    initialize_memory(blocks);

    printf("Memory Allocation Strategies Simulation:\n");

    first_fit(blocks, 30);
    best_fit(blocks, 25);
    worst_fit(blocks, 20);

    return 0;
}
```


C:\Users\kondu\OneDrive\D
Memory Allocation Strategies Simulation:
First Fit: Failed to allocate memory for size 30
Best Fit: Failed to allocate memory for size 25
Worst Fit: Failed to allocate memory for size 20

Process exited after 0.05091 seconds with return value 0
Press any key to continue . . .
29°C
Partly cloudy
Search
ENG
IN
20:22
03-03-2024