Practical 12

Natya Vidhan Biswas

B.Sc. (H) C.S. 25771

```
class MemoryBlock:
    def __init__(self, size):
        self.size = size
        self.is_allocated = False

    def get_size(self):
        return f"Allocated (Size: {self.size})" if self.is_allocated else
f"Free (Size: {self.size})"


def display_memory_blocks(blocks):
    print("\nMemory Blocks Status:")
    for i, block in enumerate(blocks):
        print(f"Block {i + 1}: {block.get_size()}")


def split_block(blocks, index, request_size):
    block = blocks[index]
    remaining_size = block.size - request_size
    block.size = request_size
    block.is_allocated = True

    if remaining_size >= 1:
        new_block = MemoryBlock(remaining_size)
        blocks.insert(index + 1, new_block)
        print(f"Block {index + 1} split. Remaining {remaining_size} bytes is
free.")


def first_fit(blocks, request_size):
    for i in range(len(blocks)):
        if not blocks[i].is_allocated and blocks[i].size >= request_size:
            split_block(blocks, i, request_size)
            print(f"\nFirst Fit: Allocated {request_size} to Block {i + 1}")
            return
    print(f"\nFirst Fit: No suitable block found for {request_size}")


def best_fit(blocks, request_size):
    best_index = -1
    min_diff = 100000000

    for i in range(len(blocks)):
        if not blocks[i].is_allocated and blocks[i].size >= request_size:
            diff = blocks[i].size - request_size
            if diff < min_diff:
                min_diff = diff
                best_index = i
    if best_index != -1:
        split_block(blocks, best_index, request_size)
        print(f"\nBest Fit: Allocated {request_size} to Block {best_index +
1}")
    else:
        print(f"\nBest Fit: No suitable block found for {request_size}")
```

```
def worst_fit(blocks, request_size):
    worst_index = -1
    max_diff = -1

    for i in range(len(blocks)):
        if not blocks[i].is_allocated and blocks[i].size >= request_size:
            diff = blocks[i].size - request_size
            if diff > max_diff:
                max_diff = diff
                worst_index = i

    if worst_index != -1:
        split_block(blocks, worst_index, request_size)
        print(f"\nWorst Fit: Allocated {request_size} to Block {worst_index +
1}")
    else:
        print(f"\nWorst Fit: No suitable block found for {request_size}")
def main():
    num_blocks = int(input("Enter the number of memory blocks: "))
    blocks = []

    for i in range(num_blocks):
        size = int(input(f"Enter size of memory block {i + 1}: "))
        blocks.append(MemoryBlock(size))

    while True:
        print("\nChoose allocation strategy:")
        print("1. First Fit")
        print("2. Best Fit")
        print("3. Worst Fit")
        print("4. Display Memory Blocks")
        print("5. Exit")
        choice = int(input("Enter your choice: "))

        if choice == 1:
            request_size = int(input("Enter the size of the memory request: "))
            first_fit(blocks, request_size)
        elif choice == 2:
            request_size = int(input("Enter the size of the memory request: "))
            best_fit(blocks, request_size)
        elif choice == 3:
            request_size = int(input("Enter the size of the memory request: "))
            worst_fit(blocks, request_size)
        elif choice == 4:
            display_memory_blocks(blocks)
        elif choice == 5:
            break
        else:
            print("Invalid choice! Try again.")
if __name__ == "__main__":
    main()
```

```
PS D:\obsidian\College\sem 3\OS\practicals> python p12.py
Enter the number of memory blocks: 5
Enter size of memory block 1: 10
Enter size of memory block 2: 15
Enter size of memory block 3: 30
Enter size of memory block 4: 25
Enter size of memory block 5: 40

Choose allocation strategy:
1. First Fit
2. Best Fit
3. Worst Fit
4. Display Memory Blocks
5. Exit
Enter your choice: 1
Enter the size of the memory request: 5
Block 1 split. Remaining 5 bytes is free.

First Fit: Allocated 5 to Block 1

Choose allocation strategy:
1. First Fit
2. Best Fit
3. Worst Fit
4. Display Memory Blocks
5. Exit
Enter your choice: 4

Memory Blocks Status:
Block 1: Allocated (Size: 5)
Block 2: Free (Size: 5)
Block 3: Free (Size: 15)
Block 4: Free (Size: 30)
Block 5: Free (Size: 25)
Block 6: Free (Size: 40)
```

```
Choose allocation strategy:
1. First Fit
2. Best Fit
3. Worst Fit
4. Display Memory Blocks
5. Exit
Enter your choice: 2
Enter the size of the memory request: 40

Best Fit: Allocated 40 to Block 6

Choose allocation strategy:
1. First Fit
2. Best Fit
3. Worst Fit
4. Display Memory Blocks
5. Exit
Enter your choice: 4

Memory Blocks Status:
Block 1: Allocated (Size: 5)
Block 2: Free (Size: 5)
Block 3: Free (Size: 15)
Block 4: Free (Size: 30)
Block 5: Free (Size: 25)
Block 6: Allocated (Size: 40)
```

```
Choose allocation strategy:
1. First Fit
2. Best Fit
3. Worst Fit
4. Display Memory Blocks
5. Exit
Enter your choice: 3
Enter the size of the memory request: 5
Block 4 split. Remaining 25 bytes is free.

Worst Fit: Allocated 5 to Block 4

Choose allocation strategy:
1. First Fit
2. Best Fit
3. Worst Fit
4. Display Memory Blocks
5. Exit
Enter your choice: 4

Memory Blocks Status:
Block 1: Allocated (Size: 5)
Block 2: Free (Size: 5)
Block 3: Free (Size: 15)
Block 4: Allocated (Size: 5)
Block 5: Free (Size: 25)
Block 6: Free (Size: 25)
Block 7: Allocated (Size: 40)
```