Assignment Number 06

Name: Mihir Unmesh Patil

Roll NO: TYCOC213

Batch: C/C-3

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
typedef struct MemoryBlock {
    int start_address;
    int size;
    bool is_free;
    struct MemoryBlock* next;
} MemoryBlock;
MemoryBlock* next_fit_pointer = NULL;
MemoryBlock* createMemoryBlock(int start_address, int size, bool is_free) {
    MemoryBlock* block = (MemoryBlock*)malloc(sizeof(MemoryBlock));
    block->start_address = start_address;
    block->size = size;
    block->is_free = is_free;
    block->next = NULL;
    return block;
}
void updateStartAddresses(MemoryBlock* head) {
    int address = 0;
    while (head) {
        head->start_address = address;
        address += head->size;
        head = head->next;
    }
}
void displayMemory(MemoryBlock* head) {
    printf("\n=== Memory Status ===\n");
    while (head) {
        printf("Start: %d | Size: %d | Free: %s\n",
            head->start_address, head->size,
            head->is_free ? "Yes" : "No");
        head = head->next;
    }
    printf("====================\n");
}
int allocateMemory(MemoryBlock** head, int process_size, char method) {
    MemoryBlock *selected = NULL, *current = *head, *start = NULL;
    int selected_size = (method == 'B') ? __INT_MAX__ : -1;
    if (method == 'N') {
        if (!next_fit_pointer) next_fit_pointer = *head;
        start = current = next_fit_pointer;
    }
    bool looped = false;
    do {
        if (current->is_free && current->size >= process_size) {
```

```
        if (method == 'F') {
            selected = current;
            break;
        } else if (method == 'B' && current->size < selected_size) {
            selected = current;
            selected_size = current->size;
        } else if (method == 'W' && current->size > selected_size) {
            selected = current;
            selected_size = current->size;
        } else if (method == 'N') {
            selected = current;
            break;
        }
    }
    current = (method == 'N') ?
            (current->next ? current->next : *head) :
            current->next;
    if (method == 'N' && current == start) looped = true;
  } while ((method != 'N' && current) || (method == 'N' && !looped));
  if (!selected) return -1;
  int start_address = selected->start_address;
  if (selected->size == process_size) {
    selected->is_free = false;
  } else {
    MemoryBlock* new_block = createMemoryBlock(0, selected->size - process_size, true);
    new_block->next = selected->next;
    selected->next = new_block;
    selected->size = process_size;
    selected->is_free = false;
    updateStartAddresses(*head);
  }
  if (method == 'N') {
    next_fit_pointer = selected->next ? selected->next : *head;
  }
  return start_address;
}
bool freeBlock(MemoryBlock** head, int start_address) {
  MemoryBlock *current = *head, *prev = NULL;
  while (current) {
    if (current->start_address == start_address) {
      current->is_free = true;
      if (current->next && current->next->is_free) {
        MemoryBlock* temp = current->next;
        current->size += temp->size;
        current->next = temp->next;
        free(temp);
      }
      if (prev && prev->is_free) {
        prev->size += current->size;
        prev->next = current->next;
        free(current);
        current = prev;
      }
      updateStartAddresses(*head);
      return true;
    }
```

```
        prev = current;
        current = current->next;
    }
    return false;
}
void deallocateAll(MemoryBlock* head) {
    while (head) {
        MemoryBlock* temp = head;
        head = head->next;
        free(temp);
    }
}
int main() {
    int total_memory, num_blocks, method_choice;
    char method_char;
    printf("Enter total memory size: ");
    scanf("%d", &total_memory);
    printf("Enter number of free blocks: ");
    scanf("%d", &num_blocks);
    int* block_sizes = (int*)malloc(num_blocks * sizeof(int));
    printf("Enter sizes of the %d free blocks:\n", num_blocks);
    for (int i = 0; i < num_blocks; i++) {
        printf("Block %d: ", i + 1);
        scanf("%d", &block_sizes[i]);
    }
    MemoryBlock *memory = NULL, *last = NULL;
    for (int i = 0; i < num_blocks; i++) {
        MemoryBlock* block = createMemoryBlock(0, block_sizes[i], true);
        if (!memory) {
            memory = block;
            last = block;
        } else {
            last->next = block;
            last = block;
        }
    }
    updateStartAddresses(memory);
    free(block_sizes);
    printf("\nChoose allocation method:\n");
    printf("1. First Fit\n2. Best Fit\n3. Worst Fit\n4. Next Fit\n");
    printf("Enter choice: ");
    scanf("%d", &method_choice);
    switch (method_choice) {
        case 1: method_char = 'F'; break;
        case 2: method_char = 'B'; break;
        case 3: method_char = 'W'; break;
        case 4: method_char = 'N'; break;
        default: printf("Invalid method\n"); return 1;
    }
    int choice;
    while (1) {
        printf("\n=== Menu ===\n");
        printf("1. Allocate process\n");
        printf("2. Free memory block\n");
        printf("3. Display memory\n");
        printf("0. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        if (choice == 0) break;
        int size, addr;
```

```
        switch (choice) {
            case 1:
                printf("Enter process size to allocate: ");
                scanf("%d", &size);
                addr = allocateMemory(&memory, size, method_char);
                if (addr == -1)
                    printf("Allocation failed: Not enough memory.\n");
                else
                    printf("Process allocated at address %d\n", addr);
                break;
            case 2:
                printf("Enter start address of block to free: ");
                scanf("%d", &addr);
                if (freeBlock(&memory, addr))
                    printf("Freed memory block at address %d\n", addr);
                else
                    printf("Invalid address or already free.\n");
                break;
            case 3:
                displayMemory(memory);
                break;
            default:
                printf("Invalid option.\n");
        }
    }
    deallocateAll(memory);
    return 0;
}
```

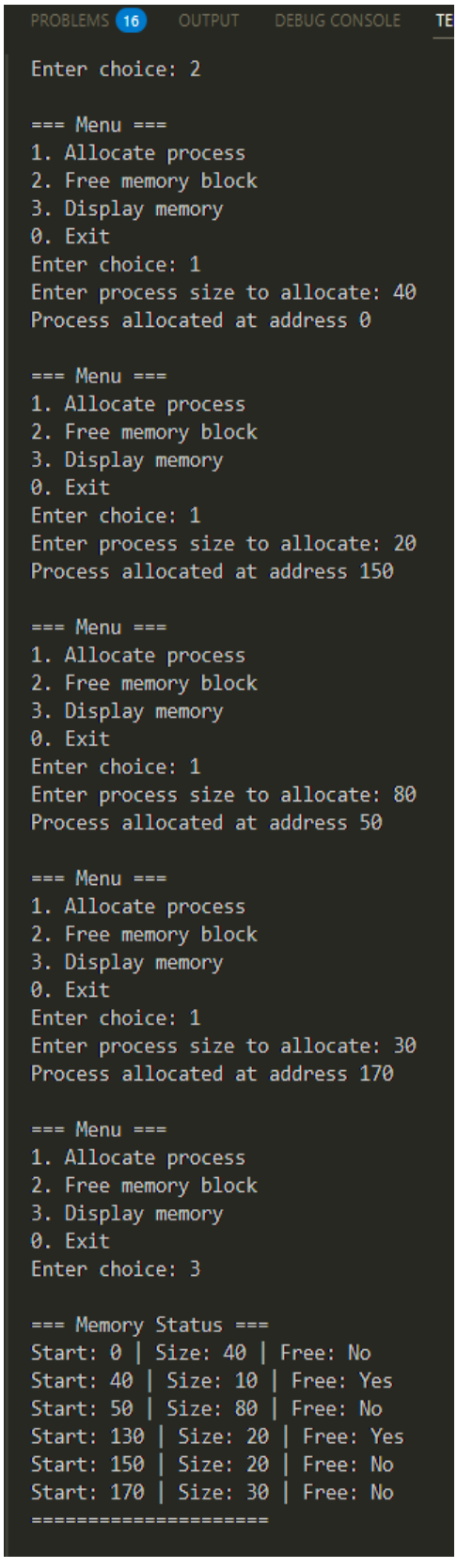
OUTPUT:

First Fit:

```
PS D:\Sem_6> cd "d:\Sem_6\OSL\Assignment_0
Enter total memory size: 200
Enter number of free blocks: 4
Enter sizes of the 4 free blocks:
Block 1: 50
Block 2: 100
Block 3: 20
Block 4: 30

Choose allocation method:
1. First Fit
2. Best Fit
3. Worst Fit
4. Next Fit
Enter choice: 1

=== Menu ===
1. Allocate process
2. Free memory block
3. Display memory
0. Exit
Enter choice: 1
Enter process size to allocate: 40
Process allocated at address 0

=== Menu ===
1. Allocate process
2. Free memory block
3. Display memory
0. Exit
Enter choice: 1
Enter process size to allocate: 80
Process allocated at address 50

=== Menu ===
1. Allocate process
2. Free memory block
3. Display memory
0. Exit
Enter choice: 1
Enter process size to allocate: 30
Process allocated at address 170

=== Menu ===
1. Allocate process
2. Free memory block
3. Display memory
0. Exit
Enter choice: 1
Enter process size to allocate: 20
Process allocated at address 130
```

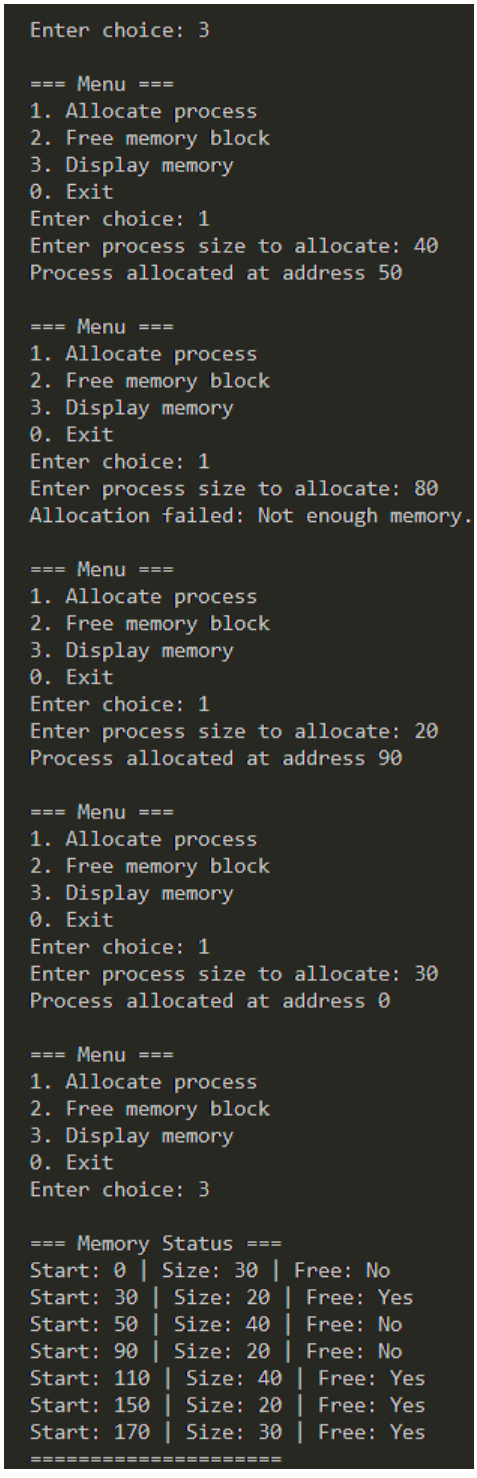
```
=== Menu ===
1. Allocate process
2. Free memory block
3. Display memory
0. Exit
Enter choice: 3

=== Memory Status ===
Start: 0 | Size: 40 | Free: No
Start: 40 | Size: 10 | Free: Yes
Start: 50 | Size: 80 | Free: No
Start: 130 | Size: 20 | Free: No
Start: 150 | Size: 20 | Free: Yes
Start: 170 | Size: 30 | Free: No
====================
```

Best Fit:



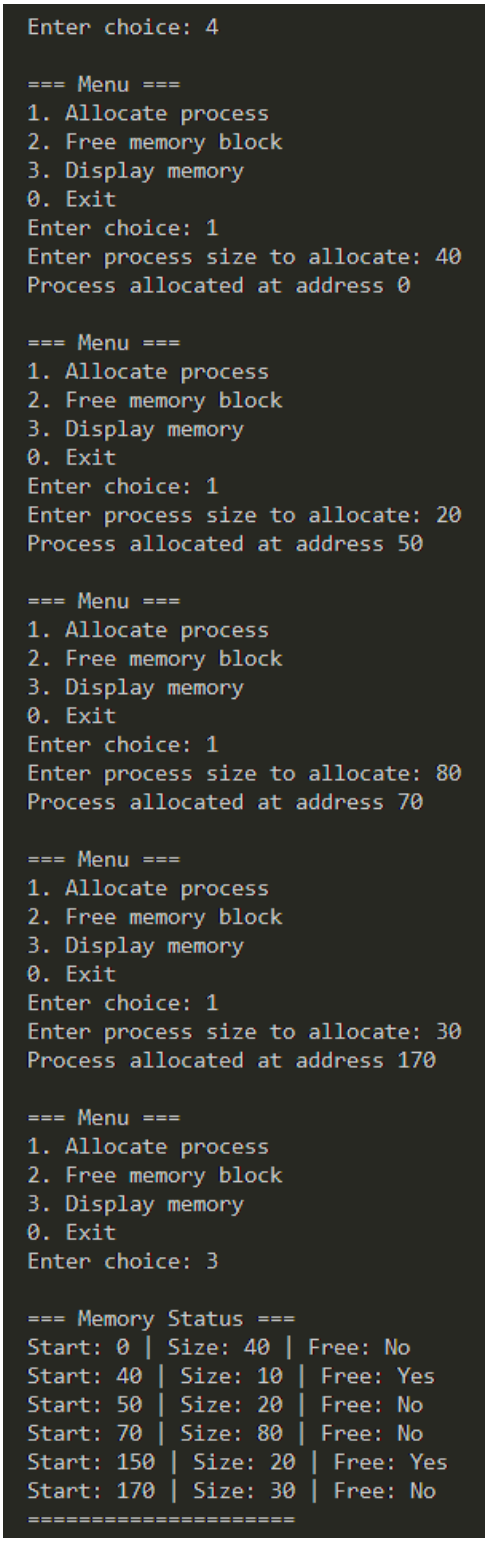
```
PROBLEMS 16  OUTPUT  DEBUG CONSOLE  TE

Enter choice: 2

=== Menu ===
1. Allocate process
2. Free memory block
3. Display memory
0. Exit
Enter choice: 1
Enter process size to allocate: 40
Process allocated at address 0

=== Menu ===
1. Allocate process
2. Free memory block
3. Display memory
0. Exit
Enter choice: 1
Enter process size to allocate: 20
Process allocated at address 150

=== Menu ===
1. Allocate process
2. Free memory block
3. Display memory
0. Exit
Enter choice: 1
Enter process size to allocate: 80
Process allocated at address 50

=== Menu ===
1. Allocate process
2. Free memory block
3. Display memory
0. Exit
Enter choice: 1
Enter process size to allocate: 30
Process allocated at address 170

=== Menu ===
1. Allocate process
2. Free memory block
3. Display memory
0. Exit
Enter choice: 3

=== Memory Status ===
Start: 0 | Size: 40 | Free: No
Start: 40 | Size: 10 | Free: Yes
Start: 50 | Size: 80 | Free: No
Start: 130 | Size: 20 | Free: Yes
Start: 150 | Size: 20 | Free: No
Start: 170 | Size: 30 | Free: No
====================
```

Worst Fit:



```
Enter choice: 3

=== Menu ===
1. Allocate process
2. Free memory block
3. Display memory
0. Exit
Enter choice: 1
Enter process size to allocate: 40
Process allocated at address 50

=== Menu ===
1. Allocate process
2. Free memory block
3. Display memory
0. Exit
Enter choice: 1
Enter process size to allocate: 80
Allocation failed: Not enough memory.

=== Menu ===
1. Allocate process
2. Free memory block
3. Display memory
0. Exit
Enter choice: 1
Enter process size to allocate: 20
Process allocated at address 90

=== Menu ===
1. Allocate process
2. Free memory block
3. Display memory
0. Exit
Enter choice: 1
Enter process size to allocate: 30
Process allocated at address 0

=== Menu ===
1. Allocate process
2. Free memory block
3. Display memory
0. Exit
Enter choice: 3

=== Memory Status ===
Start: 0 | Size: 30 | Free: No
Start: 30 | Size: 20 | Free: Yes
Start: 50 | Size: 40 | Free: No
Start: 90 | Size: 20 | Free: No
Start: 110 | Size: 40 | Free: Yes
Start: 150 | Size: 20 | Free: Yes
Start: 170 | Size: 30 | Free: Yes
====================
```

Next Fit:



```
Enter choice: 4

=== Menu ===
1. Allocate process
2. Free memory block
3. Display memory
0. Exit
Enter choice: 1
Enter process size to allocate: 40
Process allocated at address 0

=== Menu ===
1. Allocate process
2. Free memory block
3. Display memory
0. Exit
Enter choice: 1
Enter process size to allocate: 20
Process allocated at address 50

=== Menu ===
1. Allocate process
2. Free memory block
3. Display memory
0. Exit
Enter choice: 1
Enter process size to allocate: 80
Process allocated at address 70

=== Menu ===
1. Allocate process
2. Free memory block
3. Display memory
0. Exit
Enter choice: 1
Enter process size to allocate: 30
Process allocated at address 170

=== Menu ===
1. Allocate process
2. Free memory block
3. Display memory
0. Exit
Enter choice: 3

=== Memory Status ===
Start: 0 | Size: 40 | Free: No
Start: 40 | Size: 10 | Free: Yes
Start: 50 | Size: 20 | Free: No
Start: 70 | Size: 80 | Free: No
Start: 150 | Size: 20 | Free: Yes
Start: 170 | Size: 30 | Free: No
====================
```