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

typedef struct block {
    int size;
    bool free;
    struct block *next;
} *block_t;

typedef struct process {
    int pid;
    int size;
    struct process *next;
    block_t block;
} *process_t;

block_t memory;

void initialize_memory(int size) {
    memory = (block_t)malloc(sizeof(struct block));
    memory->size = size;
    memory->free = true;
    memory->next = NULL;
}

process_t first_fit_allocate(process_t process_list, int pid, int size) {
    process_t process = (process_t)malloc(sizeof(struct process));
    process->pid = pid;
    process->size = size;
    process->next = NULL;
```

```
    process->block = NULL;

    block_t curr_block = memory;
    while (curr_block != NULL) {
        if (curr_block->free && curr_block->size >= size) {
            curr_block->free = false;
            process->block = curr_block;
            break;
        }
        curr_block = curr_block->next;
    }

    if (process->block == NULL) {
        printf("Unable to allocate memory for process with pid %d and size %d\n", pid, size);
        free(process);
        return process_list;
    }

    if (process_list == NULL) {
        return process;
    }

    process_t curr = process_list;
    while (curr->next != NULL) {
        curr = curr->next;
    }
    curr->next = process;

    return process_list;
}
```

```
process_t deallocate(process_t process_list, int pid) {
    process_t curr = process_list;
    process_t prev = NULL;

    while (curr != NULL && curr->pid != pid) {
        prev = curr;
        curr = curr->next;
    }

    if (curr == NULL) {
        printf("Unable to deallocate process with pid %d as it does not exist\n", pid);
        return process_list;
    }

    curr->block->free = true;

    if (prev == NULL) {
        process_t new_head = curr->next;
        free(curr);
        return new_head;
    }

    prev->next = curr->next;
    free(curr);

    return process_list;
}

void print_memory_map() {
    printf("Memory map:\n");
```

```
    block_t curr_block = memory;
    int num_free_blocks = 0;
    int total_free_memory = 0;
    while (curr_block != NULL) {
        printf("Block size: %d, Free: %d\n", curr_block->size, curr_block->free);
        if (curr_block->free) {
            num_free_blocks++;
            total_free_memory += curr_block->size;
        }
        curr_block = curr_block->next;
    }
    if (num_free_blocks > 0) {
        printf("Average fragmentation: %f\n", (float)total_free_memory / (float)num_free_blocks);
    } else {
        printf("No fragmentation\n");
    }
}
void print_memory_map_with_wasted() {
    block_t curr_block = memory;
    int total_wasted = 0;
    while (curr_block != NULL) {
        if (curr_block->free) {
            total_wasted += curr_block->size;
        }
        curr_block = curr_block->next;
    }

    printf("Total wasted memory: %d\n", total_wasted);
}
```

```
int main() {
    initialize_memory(1024);

    process_t process_list = NULL;

    process_list = first_fit_allocate(process_list, 1, 256);
    process_list = first_fit_allocate(process_list, 2, 512);
    process_list = first_fit_allocate(process_list, 3, 128);

    print_memory_map();
    print_memory_map_with_wasted();

    process_list = deallocate(process_list, 2);
    process_list = first_fit_allocate(process_list, 4, 128);

    print_memory_map();
    print_memory_map_with_wasted();

    process_list = deallocate(process_list, 1);
    process_list = deallocate(process_list, 3);
    process_list = deallocate(process_list, 4);

    print_memory_map();
    print_memory_map_with_wasted();

    return 0;
}
```