# Memory Manager

This is a simple memory manager implementation that allows dynamic allocation and deallocation of memory blocks within a predefined memory pool. It includes both dynamic allocation and fixed allocation modes for testing purposes.

## Compilation Instructions

1. Open a terminal window.

2. Navigate to the directory containing the source code files.

3. Compile the source files using a C compiler (e.g., gcc).

```
gcc -o memory_manager_test memory_manager.c test.c
```

## How to Run the Program

1. After compilation, an executable file named `memory_manager_test` will be created in the same directory.

2. Run the executable from the terminal.

```
./memory_manager_test
```

3. The program will execute various allocation and deallocation operations based on the selected allocation mode (dynamic or fixed) and display the results.

## How to Test the Program

The program includes both dynamic allocation and fixed allocation modes for testing. The program will perform a series of allocation and deallocation operations and display the results. Here's how you can test the program:

1. Choose the allocation mode by setting the `IS_DYNAMIC_ALLOCATION` macro in the `test.c` file. Set it to `1` for dynamic allocation and `0` for fixed allocation.

2. Compile and run the program following the compilation and running instructions mentioned above.

3. The program will execute a sequence of allocation and deallocation operations and display the outcome, including the number of successful and failed operations, average times, and memory usage metrics.

4. Observe the results to ensure that the memory manager is working as expected. You can customize the `POOL_SIZE`, `MIN_ALLOCATION_SIZE`, `MAX_ALLOCATION_SIZE`, `MIN_NUM_ALLOC_DEALLOC`, and `MAX_NUM_ALLOC_DEALLOC` macros in the `test.c` file to control the testing parameters.

## Notes

- The program uses the `windows.h` library to measure time in microseconds. If you are running the program on a different platform, you may need to adjust the timing mechanism.

- Make sure to free the allocated memory pool by calling `free(manager.memoryPool);` at the end of the program to avoid memory leaks.

- Feel free to explore the `memory_manager.c` and `memory_manager.h` files to understand the internal workings of the memory manager implementation.